NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ADAPTATION OF MCORTEX TO
THE AEGIS SIMULATION ENVIRONMENT

by

Willis R. Rowe

June 1984

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Adaptation of MCORTEX to the AEGIS Simulation Environment		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Willis Ralph Rowe		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		12. REPORT DATE June 1984
		13. NUMBER OF PAGES 139
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Operating System; Microcomputers; Real-time processing; Multiprocessing; Distributed Computer Networks

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis presents the adaptation of a multi-computer real-time executive, MCORTEX, to a target environment consisting of a set of INTEL 86/12A single board computers in a MULTIBUS back plane. CP/M-86 is brought under the control of MCORTEX, and mechanisms are implemented to provide access to the MCORTEX supervisor from Digital Research's PL/I-86 language system.
Initially CP/M-86 is operating the system of micro-computers in a multi-user mode. MCORTEX and user processes are loaded from

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

CP/M-86 files. Use of all CP/M-86 functions is retained and MCORTEX can be used by PL/I-86 compiled applications programs to do multi-processing.

Approved for public release; Distribution unlimited

Adaptation of MCORTEX to the AEGIS Simulation Environment

by

Willis R. Rowe
Lieutenant, United States Navy
B.S., University of Kansas, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June, 1984

ABSTRACT

This thesis presents the adaptation of a multi-computer real-time executive, MCORTEX, to a target environment consisting of a set of INTEL 86/12A single board computers in a MULTIBUS back plane. CP/M-86 is brought under the control of MCORTEX, and mechanisms are implemented to provide access to the MCORTEX supervisor from Digital Research's PL/I-86 language system.

Initially CP/M-86 is operating the system of micro-computers in a multi-user mode. MCORTEX and user processes are loaded from CP/M-86 files. Use of all CP/M-86 functions is retained and MCORTEX can be used by PL/I-86 compiled applications programs to do multi-processing.

DISCLAIMER

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis will be listed below, following the firm holding the trademark:

1. INTEL Corporation, Santa Clara, California

INTEL	MULTIBUS
iSBC 86/12A	INTELLEC MDS
ISIS-II	PL/M-86
8086	

2. Digital Research, Pacific Grove, California

CP/M-86	LINK-86
PL/I-86	ASM-86
DDT-86	

TABLE OF CONTENTS

I. INTRODUCTION 10

A. GENERAL DISCUSSION 10

B. BACKGROUND 11

C. STRUCTURE OF THE THESIS 13

II. IMPLEMENTATION MODIFICATION ISSUES 15

A. DESIGN CONSIDERATIONS 15

B. SHARED RESOURCES 15

C. PROCESS INTEGRITY 18

D. INTERPROCESS SYNCHRONIZATION 18

E. DELETED FUNCTIONS 21

III. SYSTEM ARCHITECTURE 22

A. SYSTEM HARDWARE 22

B. OPERATING SYSTEMS 25

C. USER PROCESSES 26

IV. MCORTEX LOADER 32

A. KORE.OPS / KORE.TRC 32

B. KORE AS CMD FILE 33

C. OPERATION OF THE MCORTEX LOADER 34

V. PL/I-86 COMPATIBILITY. 36

A. THE SUPERVISOR 36

B. PL/I-86 PARAMETER PASSING CONVENTIONS . . . 38

C. PL/M REENTRANT PARAMETER PASSING 38

D. GENERATING MCORTEX PROCESSES USING PL/I-86 . 42

VI. CONCLUSIONS 47

APPENDIX A ISIS-II TO CP/M-86 KORE TRANSFER PROCEDURE . 50
APPENDIX B MCORTEX UNDER DDT86 55
APPENDIX C MCORTEX LOADER SOURCE CODE 57
APPENDIX D GATE MODULE SOURCE CODE 68
APPENDIX E DEMONSTRATION PROGRAM SOURCE CODE 75
APPENDIX F LINK86 INPUT OPTION FILES 80
APPENDIX G LEVEL II MCORTEX SOURCE CODE 83
APPENDIX H LEVEL I MCORTEX SOURCE CODE 106
APPENDIX I SCHEDULER & INTERRUPT HANDLER SOURCE CODE . 128
APPENDIX J GLOBAL DATA BASE AND INITIAL PROCESS CODE . 133
LIST OF REFERENCES 137
INITIAL DISTRIBUTION LIST 138

LIST OF FIGURES

1. IMPLEMENTATION ENVIRONMENT 23

2. MEMORY ALLOCATION 27

3. PL/I-86 MODULES 29

4. PL/I-86 PARAMETER PASSING 39

5. PL/M REENTRANT PARAMETER PASSING 41

LIST OF TABLES

1. GLOBAL MEMORY 16

2. MAP FILE 44

I. INTRODUCTION

A. GENERAL DISCUSSION

This thesis presents the adaptation of a kernel, real-time micro-computer based multi-processor operating system, called MCORTEX, to allow simultaneous user access to the CP/M operating system as well as to MCORTEX. User program development using Digital Research's PL/I-86 language system is supported.

Improvement in micro-processor capabilities, and performance, combined with continued reductions in hardware cost portend the development of powerful, relatively inexpensive micro-processor systems. Continued success in VLSI technology applications in parallel with development of appropriate operating systems will produce systems superior in many respects to computers developed using current mainframe technology. Systems of processors allow for graceful degradation under fault conditions and for distribution of the system, enhancing survivability in hostile environments. Further, parallel processing allows increased throughput and response time, and in real time application can guarantee successful monitoring at high sample rates and densities, without conflict.

A successful multi-processor system must control sequencing of inter-independent processes and access to

limited resources. For efficiency it must provide the context switching necessary for multi-processing on individual processors. Additionally, conflicts arising from simultaneous multi-processor access to common memory must be minimized without degrading throughput. This should be accomplished at a reasonable cost and in a manner that allows as many processors as are necessary to achieve the desired degree of concurrence and robustness.

The purpose of this thesis is to advance the development of a real time multi-processor system within the overall goals of the AEGIS weapons system simulation project. These goals include the demonstration of the operating system on commercially available, inexpensive, general purpose micro-computers. The system should require minimum development of both hardware and software. To the maximum extent possible, custom developments should be completely general in nature. In pursuit of these goals, MCORTEX is configured to execute in conjunction with a commercially available operating system, making the functions of both systems available to user programs. Additionally, mechanisms allowing user program development within the framework of a commercially available language system are provided.

B. BACKGROUND

The AEGIS weapons system relies on the four-processor AN/UYK-7 mainframe computer for real-time processing of large amounts of data concerning target detection and

acquisition. A project at the Naval Postgraduate School seeks to demonstrate that a system as complex as AEGIS can be controlled more economically, with improvements in graceful degradation characteristics, and without performance loss using a distributed system of micro-computers. The project requires identification and implementation of an applicable hardware configuration, development of a suitable operating system, duplication of significant real-time functions of the AEGIS weapons system and incorporation of valid simulation processes for test and evaluation of the total system.

The INTEL iSBC 86/12A, a single board micro-computer based on the 16 bit INTEL 8086 micro-processor, was selected as the system hardware base. Initial design of an operating system specific to the INTEL iSBC 86/12A was completed in 1980 and implementation was accomplished in three Naval Postgraduate School thesis in 1981 and 1982. The second thesis in this series written by Cox [Ref. 1] simplified the design of MCORTEX to more successfully address security and overhead issues in the real-time embedded applications targetted by the project. Cox also added a supervisory layer to the architecture, simplifying access and enhancing security. Klinefelter [Ref. 2] expanded and generalized Cox's work. All implementation to this point was done on the ISIS-II development system, with multi-processor test

and execution accomplished via download through a serial link to the target hardware.

The goals of this thesis are to:

1. Bring the powerful, highly portable functions of the CP/M-86 operating system under the control of MCORTEX. This will provide rapid expansion of user capabilities within the restrictions imposed by the non-reentrancy of CP/M-86 utilities. Using MCORTEX functions, control of access to CP/M-86 can be selectively applied depending on the contextual requirements of the application.

2. Sever the link with the development system, and provide a simple, convenient method of creating the MCORTEX environment. This should include user program and MCORTEX loading, transfer of control to MCORTEX, and mechanisms for return of control to CP/M-86.

3. Provide access mechanisms to the MCORTEX supervisor compatible with Digital Research's PL/I-86 language system, allowing user programs to be developed in a high level, portable language.

C. STRUCTURE OF THE THESIS

Chapter I discusses the overall direction of the AEGIS weapons system simulation project and the place this thesis holds in accomplishing project goals.

Chapter II addresses the issues which resulted in changes to MCORTEX as implemented by Klinefelter, and presents an overview of the MCORTEX functions

retained.Chapter III details the architecture of the MCORTEX environment, highlighting interactiors between the hardware, CP/M-86 and MCORTEX.

Chapter IV presents the MCORTEX loader, discussing considerations given to alternative methods for invoking MCORTEX.

Chapter V explains the interface provided between PL/I-86 and the MCORTEX supervisor. Procedures necessary to successfully create MCORTEX virtual processors are discussed.

Chapter VI summarize the current state of the system, points out problem areas, and makes suggestions for future research and testing.

II. IMPLEMENTATION MODIFICATION ISSUES

A. DESIGN CONSIDERATIONS

In a real-time system, multi-processing on a single processor decreases processor idle time. A multi-processor configuration extends the range of this economy and provides opportunities to exploit parallel and pipeline processing techniques that further enhance overall system goals. Careful consideration must be given to control of shared resources, process integrity, interprocess synchronization, methods of context switch initiation, and context switching overhead.

B. SHARED RESOURCES

The most important shared resource in a multi-processor environment is common memory. MCORTEX relies on a hierarchical bus structure to limit the requirement for access to common memory. Each processor has local memory, addressable without access to a shared bus. A process executing in local memory makes demands on the common bus only to pass computed data to external processes, or when MCORTEX functions are used. Related processes with high intercommunication rates should reside in the local memory of a single processor, thus avoiding high common bus usage.

To perform its functions, MCORTEX sets up a section of common memory called GLOBAL memory. Table 1 shows the

logical organization of this shared resource (see the last four pages of Appendix H for actual locations.). Access to

TABLE 1: GLOBAL MEMORY

OFFSET	MNEMONIC	TYPE	INIT	REMARKS
Ø	GLOBAL$LOCK	B	Ø	
1	NR$RPS	B	Ø	Number of real processors
2	NR$VPS(MAX$CPU)	B	Ø	Number of virtual processors (one byte for each possible CPU, MAX$CPU currently = 1Ø)
12	HDWINTFLAG(MAX$CPU)	B	X	Hardware interrupt flag (one for each possible CPU, MAX$CPU currently = 1Ø)
22	EVENTS	B	1	Number of events
	EVC$TBL(1ØØ)	S		Event count table
23	EVC$NAME	B	FE	Event count name
24	VALUE	W	Ø	Event count value
26	THREAD	B	FF	Event count thread
423	CPU$INIT	B	Ø	Log in CPU number
424	SEQUENCERS	B	Ø	Number of sequencers
	SEQ$TABLE(1ØØ)	S		Sequencer table
425	SEQ$NAME	B	X	Name of sequencer
426	SEQ$VALUE	W	X	Value of sequencer
	VPM(MAX$CPU * MAX$VPMS$CPU)	S		Virtual processor map (MAX$CPU currently = 1Ø, MAX$VPMS$CPU currently = 1Ø)
725	VP$ID	B	X	Virtual processor ident.
726	VP$STATE	B	X	Virtual processor state
727	VP$PRIORITY	B	X	Virtual processor priority
728	EVCAWVALUE	W	X	Count awaited
730	SP$REG	W	X	Stack pointer register
732	SS$REG	W	X	Stack segment register
1725				

B - byte W - word S - structure X - not initialized

GLOBAL memory is controlled through the combination of a hardware bus lock, and a software lock (GLOBAL$LOCK) located in GLOBAL memory. When a process sets the hardware bus

lock, it is given sole access to the common bus for one instruction cycle. During this cycle, the process makes an exchange of the value in a register (contents 77H) with GLOBAL$LOCK. The processor then examines the contents of the exchange register. If the register now contains zero, the processor is granted access, if not, the process repeats the procedure until a zero is obtained from GLOBAL$LOCK. Because access to GLOBAL memory is controlled by MCORTEX, waits should be infrequent and short in duration. When relinquishing the software lock, the process merely sets GLOBAL$LOCK to zero.

Users have no access to GLOBAL memory, however MCORTEX provides for user control of shared resources through data held in GLOBAL memory. Sequencers, located in the sequencer table section of GLOBAL memory, are used to provide a turn taking mechanism. Each shared resource is assigned a corresponding sequencer. When processes require a resource, they request a turn through the supervisory function call TICKET, specifying the applicable sequencer. TICKET returns a number indicating the callers turn at the required resource. This is similar to getting a turn number at a barber shop. TICKET advances the sequencer value in global memory so that succeeding requests receive higher numbers. The process requesting the resource then makes another supervisory call, this time on AWAIT, providing both an identification of the resource and the process turn number.

If the resource is not busy, the process will receive immediate access, otherwise the process gives up the CPU.

C. PROCESS INTEGRITY

The design of MCORTEX relies heavily on user cooperation for process integrity. The supervisor controls access to the MCORTEX functions, but even this is a software control and will not withstand malicious assult or catastrophic failure. MCORTEX is targetted at embedded systems applications where malicious assault is not expected. Protection from catastrophic failure requires hardware protection not presently in the system. The low cost of micro-computers however, allows for redundant back up systems which can limit the affects of catastrophic failure.

D. INTERPROCESS SYNCHRONIZATION

Process synchronization is accomplished under MCORTEX through the functions ADVANCE, AWAIT, and PREEMPT. These synchronizing primitives are supported with the functions CREATE$EVC, CREATE$SEQ, READ, and TICKET. Consumer processes use AWAIT to ensure that data they require is ready. Producer processes use ADVANCE to inform consumers that new data has been computed. PREEMPT is used by one process to directly ready another process. This primitive is for activation of high priority system processes of a time critical nature. A call on a synchronizing primitive may, or may not result in relinquishing the CPU. The CPU is

always assigned to the highest priority ready virtual processor on each board regardless of which synchronization function envoked the scheduler. Before using ADVANCE or AWAIT, an event count must be created using CREATE$EVC. Consumers and producers then communicate using the agreed upon event count. The current value of an event count can be determined through a call on READ. The functions of CREATE$SEQ and TICKET are as discussed earlier, but with broader applications.

MCORTEX handles two types of context switching. The first type results when control of a CPU is relinquished through a MCORTEX function call. Under these conditions the calling process is not halted in the midst of some task, but at a place 'convenient' for the process. Some subset of the processors registers contains all required state information. MCORTEX assumes this subset includes the DS, IP, CS, SS, SP, and BP registers. Additionally, a "normal" return indicator is saved. The second type of context switching results from an interrupt. This switching assumes nothing, and saves the complete state of the process being interrupted as well as an 'interrupt' return indicator. This recognition of two switch types makes context switching faster for the more common "normal" return.

Early implementers of MCORTEX considered the context switching overhead question in detail. Their solution gave greater importance to the issue of speed than to the issue

of portability. The context switching routines in MCORTEX, including the virtual processor scheduler and the interrupt handler, are the only portions of the MCORTEX core written in assembly language. Another decision motivated by the speed imperative assumed that each virtual processor owned a stack for storing state information. This decision was followed by another assuming that the stack segment pointer for each stack was different. This allowed a bootstrap like context recovery. A search through the virtual processor map identified the highest priority ready process. Virtual prossor map information included the process stack segment value. This value was retrieved, and subsequently used to retrieve three additional pieces of processor state information. Offsets zero, two, and four from the stack segment were used to retrieve the process stack pointer value, the process stack base pointer value, and a return type indicator. Recovery of the stack state allowed recovery of the entire state of the virtual processor, and processing could continue.

This context switching method has many advantages. First, once the stack segment of a process has been stored in a known, retrievable location, it never needs updating. The base of the stack remains fixed, and access is controlled through the contents of the first few bytes at the base of the stack. Second, less space is required to store the stack segment than to store the entire stack

pointer. (This information is stored in GLOBAL memory.) Third, since each process was uniquely identified by its stack segment register, MCORTEX functions did not need to identify the process they were responding to when using the scheduler. The scheduler simply stored state information at the base of whatever stack segment was active when the scheduler was called.

The assumption that each process used a different stack segment value is not completely general, and in fact was not true for procedures compiled and linked under the Digital Research PL/I-86 language system. This conflict forced changes in the context switching mechanisms of MCORTEX. The entire stack pointer (SS and SP registers) is now stored in GLOBAL memory, and MCORTEX functions making use of the scheduler must indicate (in the Processor Data Segment Table, PRDS) which virtual processor they are servicing.

E. DELETED FUNCTIONS

Functions previously available under MCORTEX include OUT$CHAR, OUT$LINE, OUTNUM, OUTDNUM, IN$CHAR, IN$NUM, and IN$DNUM. With CP/M-86 under the control of MCORTEX, these utility functions are redundant and have been removed. However a version of MCORTEX with these functions incorporated has been retained for troubleshooting purposes. The monitor process incorporated by Klinefelter has also been removed in light of the availability of DDT86.

III. SYSTEM ARCHITECTURE

A. SYSTEM HARDWARE

This implementation of MCORTEX is based on the INTEL iSBC86/12A single board computer using a MULTIBUS back plane. Specific, detailed information pertaining to both these components is available in [Ref. 3] and [Ref. 4]. The MULTIBUS also connects two memory extensions into the system. A 32K extension is used as common memory for interprocess communication under MCORTEX and for CP/M multi-user system control. A 64K extension provides additional memory required to operate the PL/I-86 compiler and other utilities not constrained to execute in the 64K of memory local to each processor. Additionally, a bubble memory system and a hard disk system are available on MULTIBUS. A second hard disk system is accessed through the parallel port of one SBC. Figure 1 is a representation of this configuration with two SBC's shown.

The iSBC86/12A provides a three level hierarchical bus structure. At the first level, the 8086 processor communicates through the on board bus with up to 4K of ROM, with serial and parallel I/O ports and with the dual-port bus. Control and access to local RAM is provided by the second level dual-port bus. The third bus level, the MULTIBUS interface, provides access to the MULTIBUS. The

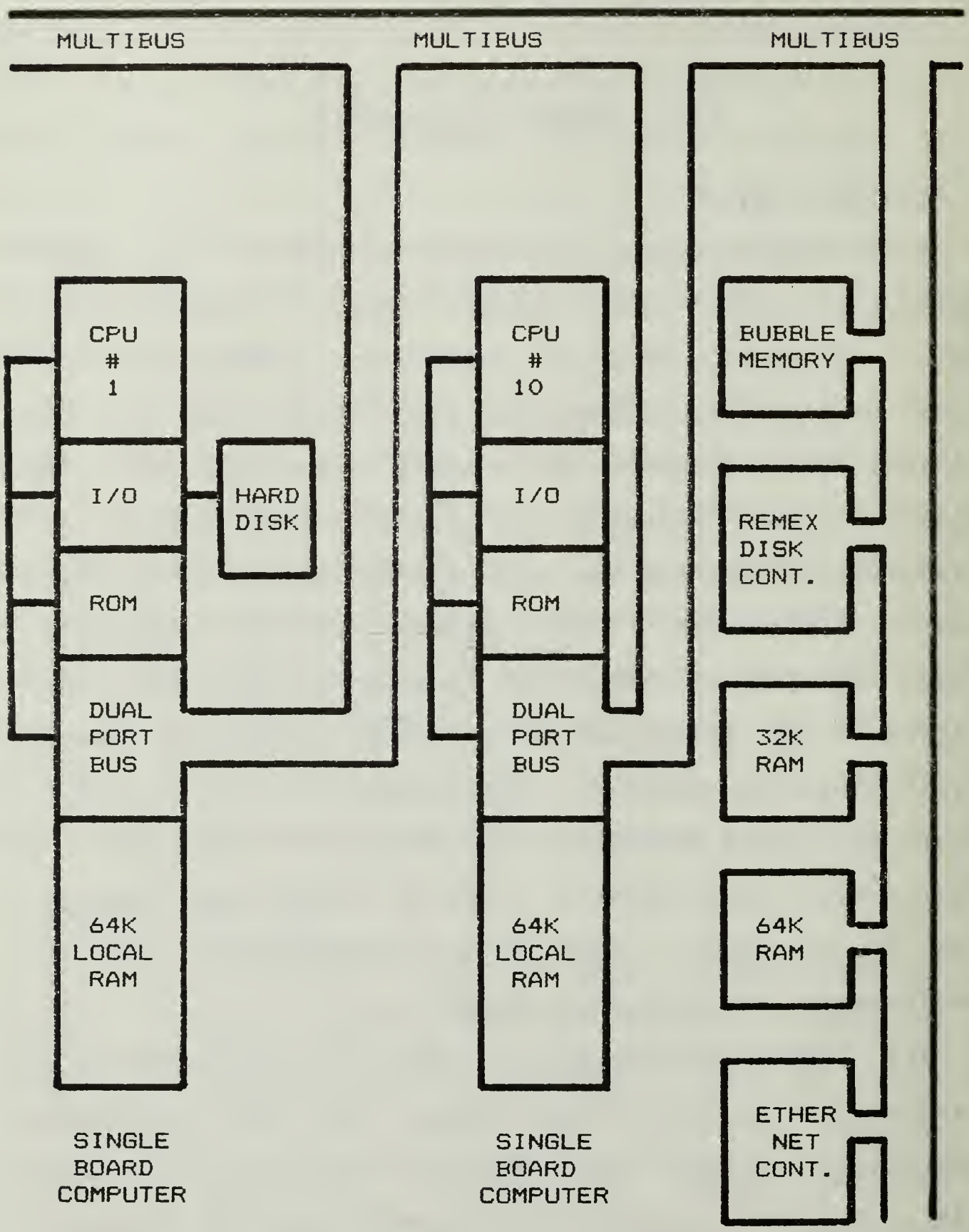

FIG. 1 IMPLEMENTATION ENVIRONMENT

presently used wiring option excludes off board access to local RAM. Differences between memory access times at the first two levels are negligible, but memory accesses involving MULTIBUS require a minimum 25% increase in access time.

The high performance, general purpose 8086 micro-processor base of the iSBC86/12A contains an Execution Unit (EU) and a Bus Interface Unit (BIU). EU functions are supported by instruction fetches and operand reads and writes conducted by the BIU. The BIU can stack instructions in an internal RAM to a level of six deep increasing EU efficiency and decreasing bus idle time.

The 8086 has eight 16 bit general purpose registers, four being byte addressable. The remaining four are primarily pointer registers, but can be used as accumulators. Additionally, the 8086 has four segment registers, an instruction pointer register and a flag register with nine status bits.

A segmented one mega-byte address space is provided for by the 8086 micro-processor. This is accomplished by combining the 16 bits of each segment register left shifted four bits, with the 16 bits of an associated pointer register unshifted. The resulting 20 bits form a physical address. For any given segment register value 64k bytes of memory can be addressed through manipulation of the pointer register alone. The 64k byte memory spaces formed can be

discrete or can overlap on boundaries that are multiples of 16 bytes, depending on segment register values.

The iSBC86/12A provides serial I/O through an INTEL 8251A USART, parallel I/O through an INTEL 8255A PPI and a broad range of interrupt control through the INTEL 8259A PIC. MCORTEX operates using interrupt 4. The interrupt is generated via output to parallel port B, as proposed by Perry [Ref. 5: pp. 65 to 69]. Both the hardware and software implementations are exactly as presented by Perry.

B. OPERATING SYSTEMS

A copy of MCORTEX resides in each processors local memory and is a distributed part of the address space of each local process. Additionally, GLOBAL memory is accessible to MCORTEX to facilitate interprocess synchronization. A system interrupt under MCORTEX control, in conjunction with interrupt flags maintained in GLOBAL memory, provides communication initiation between real processors. Upon receiving an interrupt, each processor checks its flag in GLOBAL memory to determine if the interrupt is intended for a process in its local memory. If not, the process executing at the time of the interrupt continues. Otherwise a call is made to the MCORTEX scheduler and the highest priority ready process is given control of the CPU. For communication between processes in a common local memory, no interrupt is issued, a call to the scheduler is made directly.

Access to MCORTEX is through the supervisor at the outermost layer of the MCORTEX four level structure discussed by Klinefelter [Ref. 2 : pp. 44-46]. Due to incompatible parameter passing implementations in PL/M-86, and in PL/I-86, code allowing PL/I-86 access to the MCORTEX supervisor has been developed. This is discussed fully in Chapter V.

Also resident in each local memory, if required, is the CP/M-86 operating system. In this configuration the full range of CP/M utilities, [Ref. 6] and [Ref. 7], is available to the user. Additionally, development of user processes can make use of any of the broad scope of commercially available products compatible with CP/M-86. Figure 2 gives a representation of the locations of the system code. The diagram includes the location of DDT-86 as required for a debugging session. Also depicted are the locations of the MCORTEX / MXTRACE loaders. During load, loader memory is not reserved, and care must be taken to ensure that a CMD module's code or data section does not overwrite it. It is permissible, however, to include this memory as part of a module stack or free space, since these structures are developed at module runtime when loader functions have been completed.

C. USER PROCESSES

User processes may be located in areas indicated in Figure 2. Additionally, if CP/M-86 utilities are not

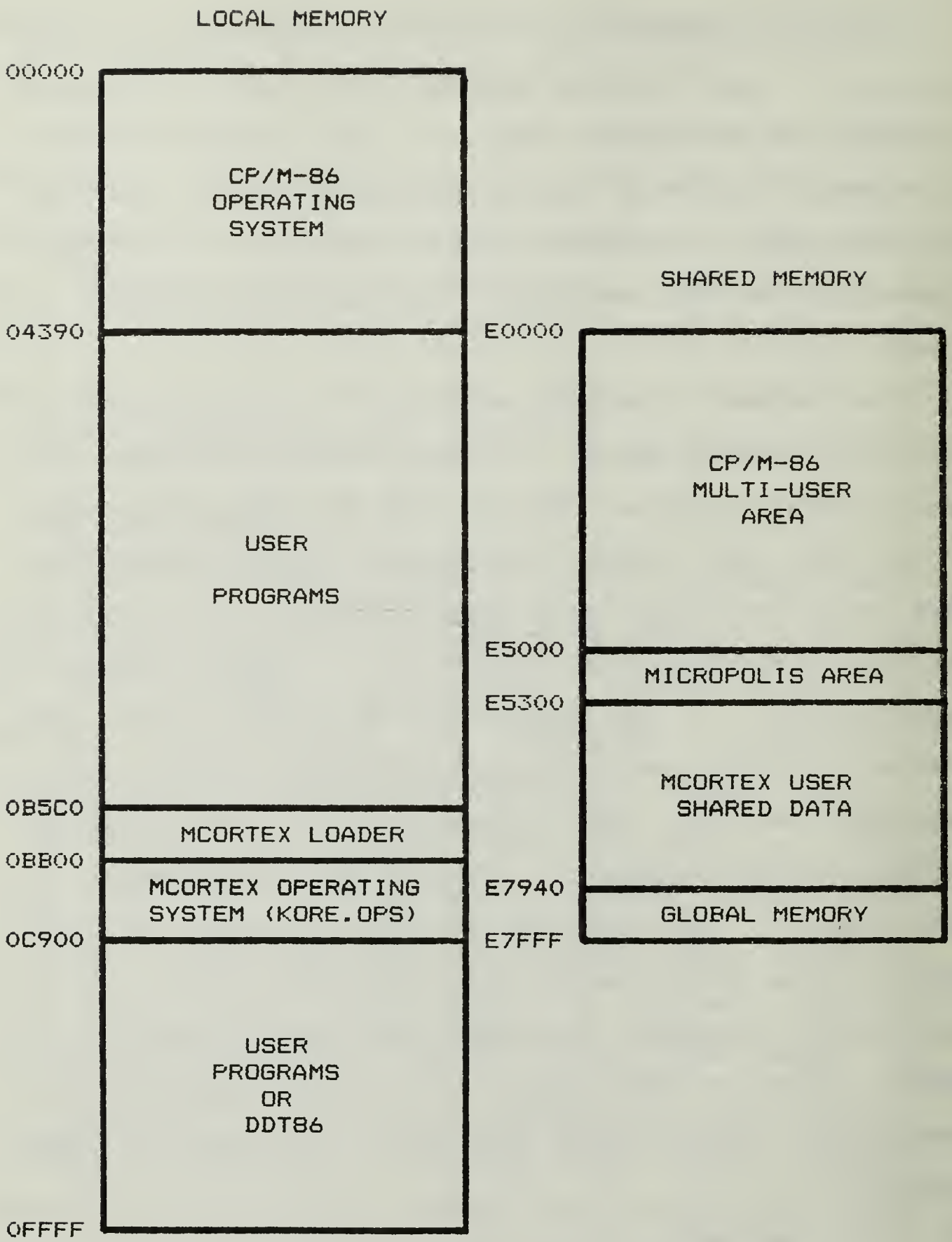

FIG. 2 MEMORY ALLOCATION

required, memory reserved for CP/M-86 may hold user processes.

Descriptions of processes in memory are provided to MCORTEX through CREATE$PROC. This MCORTEX function gives the process a unique identification number, priority, stack (SS and SP registers), next execution address (CS and IP registers), data segment (DS register), and extra segment (ES register). MCORTEX establishes the process initial context using this information to create a virtual processor. The virtual processor exists as a combination of data, both in GLOBAL memory, and in each process stack. When executing, the virtual processor becomes identical with the real processor state. Relinquishing the CPU forces the virtual processor again into GLOBAL memory and the process stack.

Special effort has been made to accommodate processes created under PL/I-86 and linked using LINK86. The internal architecture of such processes requires some consideration. LINK86 concatenates all PL/I-86 code segments into one segment. The same is done with data segments. Thus, PL/I-86 processes consist of a series of contiguous code segments followed by a series of contiguous data segments. Additionally, at run time PL/I-86 routines create a stack following the data area, and a free space following the stack. The resulting process configuration is shown in Figure 3.

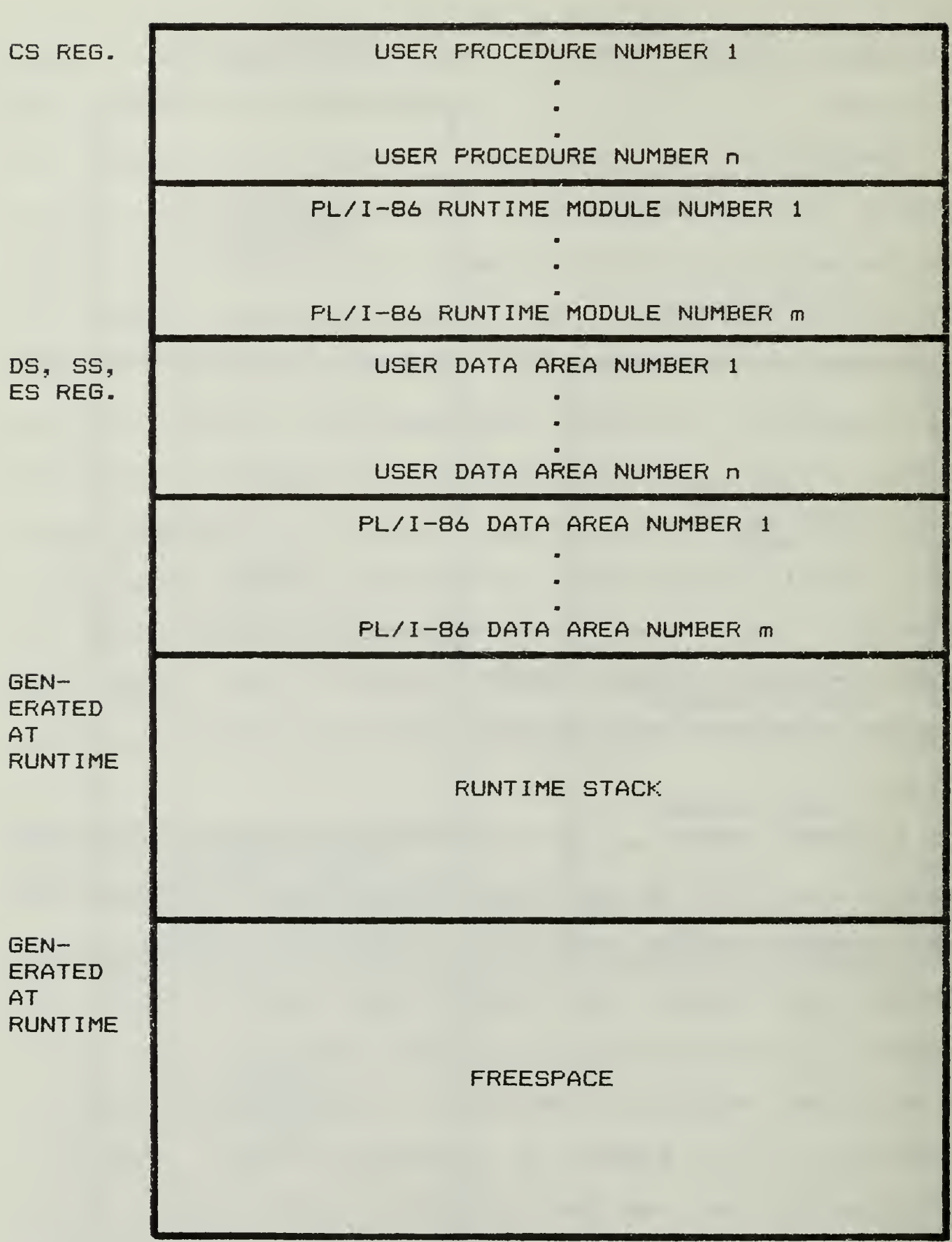

FIG. 3 PL/I-86 MODULES

Access to all data areas resulting from a single link, is referenced to a common data segment. Stack pointers are referenced to the stack segmert register, and free space pointers to the extra segment register. Additionally, some PL/I-86 runtime routines assume the contents of all three segment registers (DS, SS, ES) are identical. This assumption disallows process stacks with unique stack segments, and was the motivation for modifications to MCCRTEX discussed in Chapter II. For the demonstration programs D1.CMD and D2.CMD (Appendix E) PL/I-86 generated a default stack of size 400H bytes. This area was subdivided to provide a 120H byte process stack and a 2E0H system stack in the case of D1.CMD, and two 120H byte process stacks and a 1C0H byte system stack in the case of D2.CMD. The documentation for PL/I-86 [Ref. 8 : p. 2.9] describes mechanisms incorporated in the PROCEDURE statement to specify the size of the runtime stack. If these mechanisms function as described, all process stacks can be contained within the area allocated to the runtime stack. Otherwise process stacks can be constructed following the free space. This area would be unprotected by normal CP/M CMD file memory management functions, and its use would require extra care.

The MCORTEX CREATE$PROC parameters include the absolute location of process start, stack, and data. For this reason it is advantageous to locate processes absolutely when

linking. LINK86 provides such an option [Ref. 9 : p. 7.6], however, the ABSOLUTE option is applicable to the entire CMD file created and cannot be used to distribute the file non-contiguously in memory. Also, experience has shown that the required code segment address must be placed in the data's ABSOLUTE declaration. Further, the code segment ABSOLUTE declaration must hold an address larger than the sum of the value placed in the data ABSOLUTE declaration and the size of the data segment. This value seems to have no effect on the location of the file but, too small a value will cause an error when the file is loaded. See Appendix F for examples of link option files that produce correct results.

MCORTFX processes may be linked together as PL/I-86 procedures allowing sharing of PL/I-86 runtime routines or may be linked individually. Separate processes reouire more memory due to replication of PL/I-86 support routines, however, great care is required with shared routines as PL/I-86 runtime routines are not reentrant. Further, CP/M-86 subroutines are neither reentrant nor replicateable. I/O functions, therefore, must be viewed as shared resources and access to them strictly controlled.

IV. MCORTEX LOADER

A. KORE.OPS / KORE.TRC

During development the MCORTEX executive was assigned to the file KORE and was accessible through the INTELLEC MDS system. This file contained all the multi-processor operating system functions, the initial GLOBAL memory, the supervisor, the interrupt vector, and various low level functions not accessible to the user. To execute MCORTEX it was necessary to download KORE and user processes to the target system, disconnect the transfer cable, connect the target system terminals, and pass control to KORE on each processor. See [Ref. 2: Appendix A, B] for a complete description of the process. The KORE.OPS and KORE.TRC files loaded by the MCORTEX and MXTRACE loaders respectively, are derived from the original KORE file with changes as discussed in Chapter II. Additional changes were made to compact the KORE.OPS file, and to relocate the INIT$MOD for simpler, more CP/M-86 compatible loading of user processes. Appendix A details the procedure used to produce KORE.OPS and KORE.TRC from KORE. Further discussion will use the terms KORE and MCORTEX to mean either KORE.OPS or KORE.TRC and MCORTEX or MXTRACE respectively. When this generalization does not hold, the differences will be noted.

Currently the MCORTEX environment can be established under the CP/M-86 operating system. Control is then passed to MCORTEX automatically, and user processes are created in the user initialization module. Control can be passed back to the CP/M-86 operating system if applicable.

B. KORE AS CMD FILE

Establishment of the MCORTEX environment through invocation of KORE as a command file is not feasible for several reasons. First, interpretation of CMD file headers assumes each CMD file to be contiguously constructed. KORE is not. Second, KORE memory requirements include an interrupt vector. The CP/M-86 memory management system does not allow loading of command files into the interrupt vector space. Third, the data segment for the initialization module depends upon the amount of executable code generated by all processes linked with the module and is not static. The data segment register initial value must be passed to KORE after processes are loaded. Fourth, KORE includes GLOBAL memory, which should be loaded only once, while KORE must be loaded into each processors local memory. An additional consideration is the simplicity and flexibility gained when KORE and user processes are loaded via the same mechanism to produce the MCORTEX environment.

C. OPERATION OF THE MCORTEX LOADER

MCORTEX.CMD is an executable file under the CP/M-86 operating system. Invocation of MCORTEX without KORE.OPS on the default drive results in an error message and return to CP/M-86. MXTRACE requires KORE.TRC. The loader announces that it is on line, and requests an entry to indicate whether or not GLOBAL memory should be loaded. Only the first processor activated should load GLOBAL memory. Subsequent loads of GLOBAL memory would destroy data needed by executing processors. If no initial load of GLOBAL memory is made the results are unpredictable.

KORE is immediately loaded with or without GLOBAL memory as directed. The load is accomplished using CP/M-86 functions, but does not use the CMD load utility. Instead, KORE is read in and positioned block at a time as required. The interrupt vector is not maintained as a part of the KORE files, but is generated within the loader itself with moves directly from loader data memory to the interrupt vector space.

KORE load is followed by a request for a process file name. The loader expects at least one file name to be entered, and results are unpredictable if one is not. User processes are loaded using the CP/M-86 CMD load utility, and user processes must be CMD files. The entire file name must be entered including the three letter extension. After loading the first and subsequent user files, the loader

requests another file name. To exit user process loading, a return with no preceding character should be entered. The last file entered must contain the initialization module, as the data segment register value of this file is determined and passed to KORE.

Completion of user process loading causes control to be passed to MCORTEX. MCORTEX initializations are performed, including creation of the IDLE and INIT processes (also MONITOR with MXTRACE), and the user initialization process is entered. Operation after this point is determined by the user processes. An ADVANCE on the initialization event count 'FE' by any process will halt all processors, returning them to CP/M-86 control. The demonstration programs in Appendix E end with a PREEMPT call to the INIT process. This is only to demonstrate the operation of PREEMPT and, in fact, due to multiple declarations of the INIT process causes only the first processor activated to return to CP/M-86 control.

V. PL/I-86 COMPATIBILITY

A. THE SUPERVISOR

KORE is written in PL/M-86, and requires calls made to the supervisor to meet PL/M-86 parameter passing conventions. Further, the supervisor requires four parameters with every call regardless of the function invoked. To meet parameter passing requirements, and to hide details of the supervisor implementation, a translation mechanism between user calls and the supervisor is required.

The first parameter expected by the supervisor is a byte value indicating the function required. Following the function code should be another byte, a word and a pointer. The formal parameters these actual parameters represent are different for different function calls, and in some cases the values passed are not used at all. The supervisor uses the function code to determine which parameters are applicable, and simply ignores the rest. It is inconvenient and unnecessary for the user to provide unneeded parameters or to remember which function codes belong to which functions.

Two files (see Appendix D) are provided to mitigate differences between simple user calls and supervisor requirements. The file GATEWAY.PLI should be %INCLUDE'd in all programs making calls on MCORTEX functions. It declares

the MCORTEX functions as ENTRY values with attribute lists matching the parameters expected by GATEMOD. Note that entry declarations reserve memory space for the parameters specified. Each user process must have separate memory set aside for these function calls to avoid concurrency problems in GATEMOD.

GATEMOD.OBJ (or GATETRC.OBJ) should be linked with all user processes. It provides the object code necessary to convert user calls to the format expected by the supervisor, including addition of function codes, and padding of calls with extraneous parameters. GATEMOD uses no variable data segment of its own, and simply makes moves from user data areas to the user stack. This ensures that, so long as the user data areas involved are unshared, GATEMOD is reentrant.

Note that all parameters in the GATEWAY declarations are BIT(8) or BIT(16). PL/M has two unsigned integer data types, BYTE and WORD, that are used extensively in MCORTEX. There are no corresponding data types in PL/I-86, and BIT(8) and BIT(16) are the closest available substitutes. In MCORTEX processes it is sometimes convenient to add two BIT(16) numbers. Unfortunately, mathematical computations on BIT(16) values are not supported in PL/I-86. This set of conditions necessitated the development of the function Add2BIT16 included in GATEWAY. As the name implies, this function adds two BIT(16) parameters as unsigned integers and returns the result as a BIT(16) value. If a carry is

produced, it is ignored, and the result returned will, of course, be incorrect.

B. PL/I-86 PARAMETER PASSING CONVENTIONS

Parameters passed in a PL/I-86 procedure call are accessed via an array of pointers [Ref. 10: p. 18.1]. The location of the pointer array is provided to called routines through a pointer in the BX register. Using register indirection and indexing, pointers to actual parameters are loaded into system pointer registers. Parameter values can then be manipulated as required. Figure 4 is a diagrammatical representation of the parameter passing structure that might be established by PL/I-86 for a call on the MCORTEX supervisor.

All BIT(16) values returned to user programs by the GATEMOD, either as a result of a call to ADD2BIT16 or as a result of calls to the MCORTEX functions READ or TICKET, are returned in the BX register. This is the convention followed by 8086 based PL/I-86.

C. PL/M REENTRANT PARAMETER PASSING

All MCORTEX PL/M-86 routines are reentrant. The ASM86 routines lock out interrupts during execution so that reentrancy is not an issue. In particular the MCORTEX supervisor is reentrant. This is the only KORE module accessible to user processes.

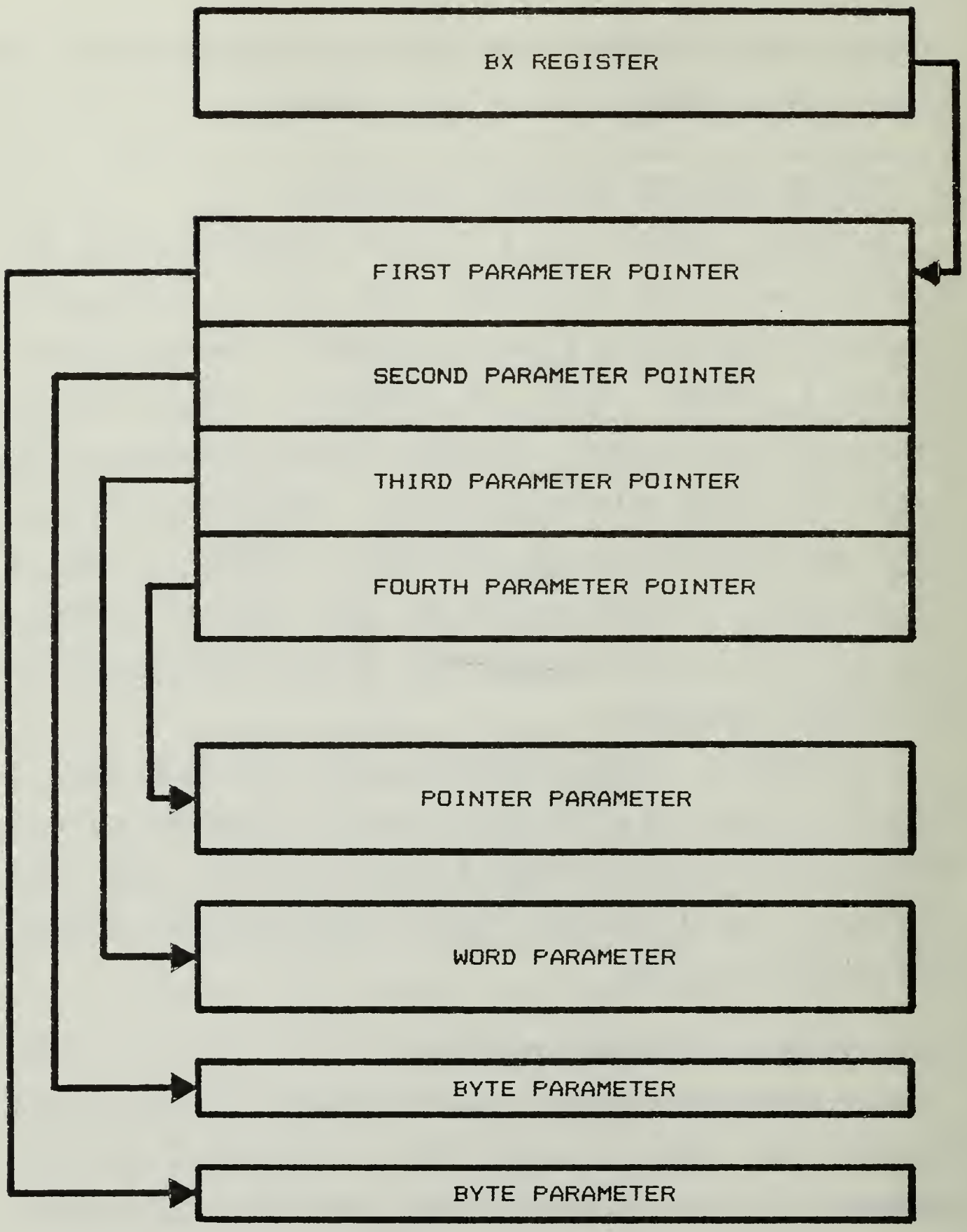

FIG. 4 PL/I-86
PARAMETER PASSING

PL/M-86 reentrant processes expect parameters to be passed on the stack in the order they appear in the procedure declaration. Byte values require two bytes on the stack even though only one byte contains usable information. Parameters are followed immediately on the stack by the call generated return address. The called process stores the callers DS and BP registers on the stack, and establishes its own DS and BP values. Access to parameters is via an index referenced to the called process BP value. Figure 5 is a diagrammatical representation of how a stack is structured following a call to GATE$KEEPER.

GATEMOD and GATETRC both act as translators of user calls into formats required by the MCORTEX and MXTRACE supervisors respectively. The only difference in the two gate modules is the address of GATE$KEEPER in their associated KOREs. Using the BX register link to retrieve data, they build the stack structure expected by the supervisor module, supplying function codes and padding when required. They then make a call on GATE$KEEPER. If the call is to READ or TICKET, space is reserved on the stack for the returned value. This value is POP'ed into the BX register before exiting to the calling process.

The gate modules provide one additional service. KORE functions do not guarantee the integrity of the ES register. PL/I-86 in OPTIONS (MAIN) initializations, however, establishes the ES, SS, and DS registers to be of equal

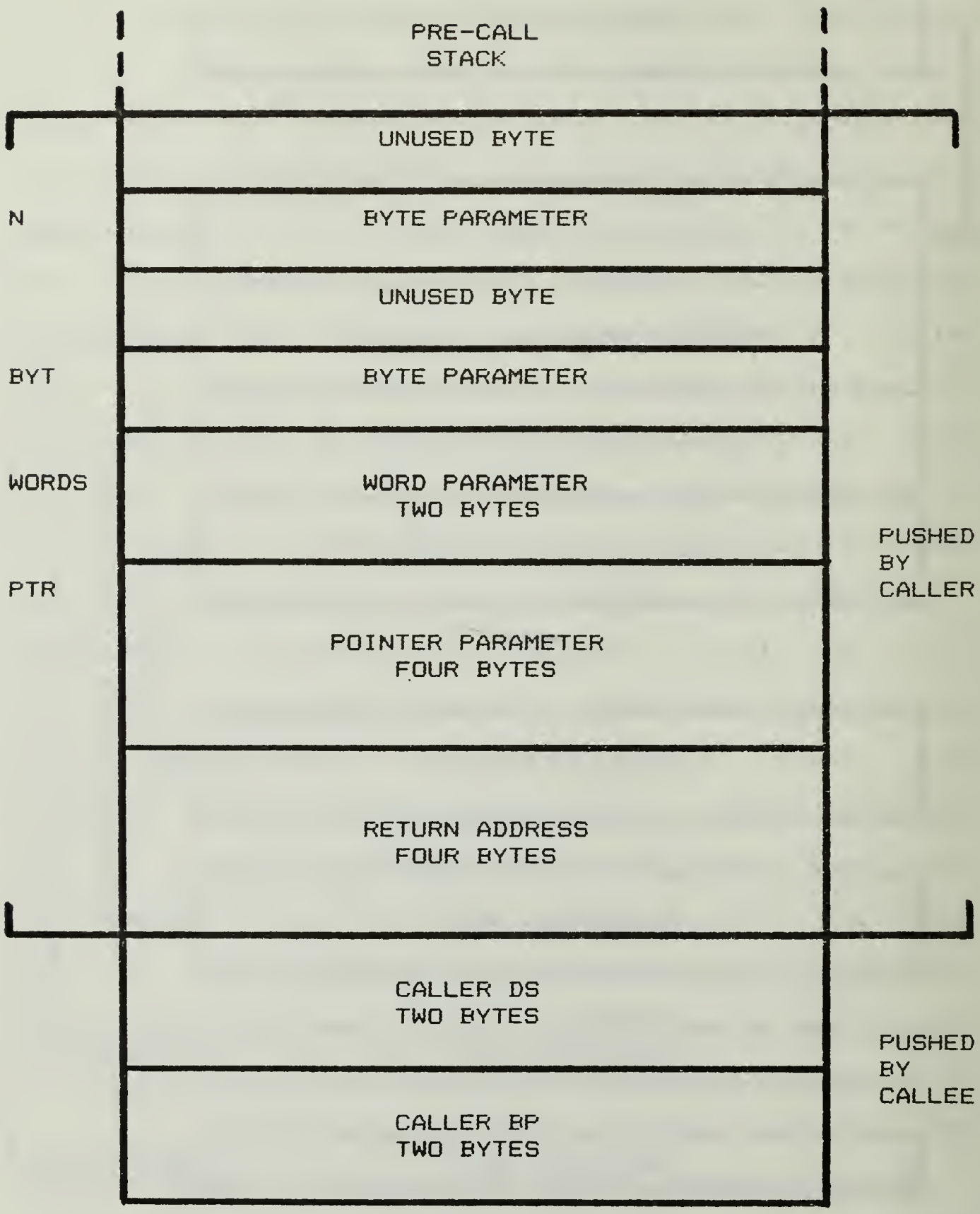

FIG. 5 PL/M REENTRANT PARAMETER PASSING

value, and some runtime routines expect this relationship to be maintained. To overcome the consequences of these opposing positions, the gate modules push the ES register onto the stack on entry, and pop it before return to the calling routine. From the standpoint of user processes, the ES register value is unchanged during MCORTEX calls.

D. GENERATING MCORTEX PROCESSES USING PL/I-86

Procedures written in PL/I-86 become MCORTEX processes via execution of CREATE PROC functions. MCORTEX processes, though written, compiled, and linked as PL/I-86 procedures, are distinct processes. Each requires the state of the processor to be prepared by the MCORTEX executive prior to every entry into the process. This is accomplished transparently when making MCORTEX function calls. Procedures in a MCORTEX process can be accessed from within the process normally, however, a MCORTEX process must be entered through a MCORTEX function call, and never through a PL/I-86 procedure call. Also MCORTEX processes can be linked into a single CMD module or can be developed as separate CMD modules. In the first case processes may share common PL/I-86 runtime routines as well as CP/M-86 utilities. In the second case PL/I-86 runtime routines are not shared, but CP/M-86 utilities, if used, are still shared.

MCORTEX currently expects an initialization module to be located starting at 04390H. This module is the first user process executed, and can be used to create user event

counts and sequencers, as well as processes. After all initializations are performed, an AWAIT('FE'B4,'0001'B4) should be executed. This puts all initialization processes on a common reserved event count thread. An ADVANCE('FE'B4) by any process will return all processors to CP/M-86 control (assuming CP/M-86 is resident locally).

MCORTEX processes are written as parameterless PL/I-86 procedures. Execution of CREATE PROC functions in the initialization module establishes a virtual processor for each process, and sets all process states to ready. The AWAIT call at the end of initializations forces a scheduling to take place. The highest priority virtual processor will be granted access to the real processor. Further scheduling is controlled by user processes using MCORTEX functions.

Parameters required by the CREATE PROC function include values unknown to the programmer until after all processes have been compiled and linked. This requires that dummy values be provided for the first compilation and linking. Links should be performed with the MAP command option selected, as this provides information relevant to user process definition. A partial MAP print out for the D1 demonstration process is shown in Table 2.

CREATE PROC has eight parameters. The first two are process identification and process priority. These are arbitrary BIT(8) values assigned by the programmer. Four other parameters, the CS, DS, SS, and ES register values,

TABLE 2:

Map for file: D1TRC.CMD

Segments

Length	Start	Stop	Align	Comb	Name	Class
2CE3	(0000:0005-2CE7)	BYTE	PUB	CODE		CODE
05C4	(0000:0100-06C3)	WORD	PUB	DATA		DATA
0021	(0000:06C4-06E4)	WORD	COM	?CONSP		DATA
0013	(0000:06E6-06F8)	WORD	COM	?FPBSTK		DATA
002E	(0000:06FA-0727)	WORD	COM	?FPB		DATA
0002	(0000:0728-0729)	WORD	COM	?CNCOL		DATA
0009	(0000:072A-0732)	WORD	COM	?FILAT		DATA
0008	(0000:0734-073B)	WORD	COM	?FMTS		DATA
001B	(0000:073C-0756)	WORD	COM	?EBUFF		DATA
0003	(0000:075E-075A)	WORD	COM	?CNCOD		DATA
0025	(0000:075C-0780)	WORD	COM	SYSIN		DATA
0028	(0000:07E2-07A9)	WORD	COM	SYSPRINT		DATA

Groups	Segments			
CGROUP	CODE			
DGROUP	DATA	?CONSP	?FPBSTK	?FPB
	?CNCOL	?FILAT	?FMTS	?EBUFF
	?ONCOD	SYSIN	SYSPRINT	

map for module: INIT

001E (0000:0005-0022) CODE
0021 (0000:0100-0120) DATA

map for module: MCDEMO

0072 (0000:0023-0094) CODE
0039 (0000:0122-015A) DATA

map for module: LOG_ON

0127 (0000:0095-01BB) CODE
00C0 (0000:015C-021B) DATA

map for module: GATEM/T

00D0 (0000:01BC-028B) CODE
0004 (0000:021C-021F) DATA

can be determined by performing an executable load of the process CMD file under DDT86. Values displayed by DDT86 include the CS, and DS register values. As mentioned earlier, it is required that the DS, SS, and ES register values be equal for proper operation of some PL/I-86 runtime routines. Except under special carefully considered circumstances, programmers should ensure that this is the case. The remaining two parameters are pointer values obtainable from the link MAP file.

The first section of the MAP file gives a summary of all code and data segments included in the associated CMD file. Several data segments are listed in order of their occurrence in memory, from lowest offset to highest offset. The range of the last entry gives the last address offset occupied by any data segment. Higher address offsets still within the memory space of this CMD file are assigned to stack and free space structures by PL/I-86, with the system stack preceding free space. The SP value required by the CREATE PROC function can be obtained by adding the size of the stack required to the last offset occupied by data. If another MCORTEX process stack is required, its SP can be obtained by adding its size to the SP of the previous process. The system stack can be divided as necessary by continuing in this manner. The total number of bytes occupied by MCORTEX process stacks should not exceed the number of bytes provided by PL/I-86 for the system stack.

The MAP file also contains maps of the individual modules linked into the CMD file. These maps provide data about locations of code and data segments within the larger code and data segments summarized in the segments section. The beginning address of each module is given. This offset represents the IP value for that particular module.

With all parameter values determined, the initialization process must be recompiled, and all processes relinked. The resulting CMD file can be executed in the MCORTEX environment.

VI. CONCLUSIONS

The principal goals of this thesis were met. MCORTEX has been integrated into a selected environment to provide multi-processing and multi-processor capabilities. Assets available under the CP/M-86 operating system have been made available to MCORTEX processes. Also, development of MCORTEX processes in the high-level language PL/I-86 has been provided for through reentrant gateway transformations between PL/I-86 calling structures and the structures expected by the MCORTEX supervisor. Programs have been written to demonstrate that each of the MCORTEX functions can be used from within a process written in PL/I-86. Two versions of the operating system kernel have been produced. One version, found in the KORE.TRC file, retains all diagnostic cues of the development version, primitive I/O functions, and the MONITOR. The second version, found in the KORE.OPS file, has these items removed.

No testing of the system, except to monitor the proper operation of the demonstration programs, has been accomplished. The demonstration programs have been run successfully using two slave SBC's and using the master SBC and one slave SBC. The loader program sometimes will not accept a file name without the drive prefix. No pattern to this behavior has been observed.

As noted earlier, neither CP/M-86 nor PL/I-86 runtime routines are reentrant. Sharing any section of code from either system in a concurrent environment requires care and proper control of access to shared code. In many instances this can be accomplished through application of MCORTEX functions. When proper sequencing through PL/I-86 runtime routines cannot be guaranteed, processes using shared routines must be separated, and multiple links performed. This produces a copy of the runtime routines for each linked group of modules. Since processes not linked into the same CMD file do not share common data structures, communication between the modules becomes more complex. PL/I-86 uses sixteen bit pointers, and has no built in mechanism to transfer data outside the data segment assigned to the linked module. This deficiency also adversely affects the communication through common memory of processes on separate SBC's.

Future research with MCORTEX should investigate the problems discussed above. Testing of the system using more than two SBC's should be conducted. Investigation of the relationships between MCORTEX processes sharing sections of PL/I-86 and CP/M-86 code must be conducted, and the best means of controlling access to shared code determined. If possible, this should be accomplished in a high priority system process that is transparent to users. Some convenient means must be provided to give processes access to memory

outside their assigned data segments. Finally, AEGIS system processes and real time input simulation processes should be adapted to the MCORTEX environment, and performance measurements made.

APPENDIX A

ISIS-II TO CP/M-86 TRANSFER

I. PRE-POWER-ON CHECKS

A. SBC configured for CP/M-86 cold boot is in MULTIBUS odd slot and no other clock master SBC is installed.

B. Bubble memory is in MULTIBUS.

C. REMEX controller is in MULTIBUS, and properly connected to REMEX drive.

D. If MICROPOLIS hard disk is to be used, ensure that it is connected to clock master SBC.

E. Ensure 32K shared memory module is installed.

F. Connect RS232 transfer cable between J2 on SBC, and 2400 baud CRT port of the MDS system. If this cable has a 'null modem' switch on it, set it to "null modem". This transposes wires 2 and 3. The switch may alternately be marked 'computer to computer' and "computer to terminal". Set to "computer to computer".

G. Connect any CRT to the 9600 baud TTY port of the MDS system. Ensure CRT is set to 9600 baud.

H. A CRT will be connected to the SBC after the loading is completed, and should have an RS232 cable hooked to the serial port. The CRT connection should lead to a flat 25 wire ribbon and J2 connector so it can eventually be hooked to the SBC's serial port.

II. POWER ON PROCEDURES

A. Turn the power-on key to ON position at MULTIBUS frame.

B. Press RESET near power-on key.

C. If needed apply power to MICROPOLIS hard disk.

D. Apply power to REMEX disk system. After system settles, put START/STOP switch in START position. Following a lengthy time-out period, the READY light on the front of the REMEX disk system will illuminate, and the system is ready. Alternately, the RESET button on the MULTIBUS can be pressed three times, with a small time-out for the system to settle each time. Following the third button push, the READY on the front of the REMEX disk system will illuminate as before.

E. Insert the boot disk into drive B.

F. Apply power to the CRT.

G. Put the Bubble Device RUN/HALT switch to RUN.

H. Power up the MDS disk drive.

I. Power up the MDS terminal.

J. Turn power-on key to ON at MDS CPU.

III. BOOT UP MDS

A. Place diskette with executable modules and SBC861 in drive 0.

B. Push upper part of boot switch in (It will remain in that position).

C. Press reset switch and then release it.

D. When the interrupt light #2 lights on the front panel, press space bar on the console device.

E. Reset the boot switch by pushing the lower part of the switch.

F. ISIS-II will announce itself and give the '-' prompt.

IV. LOAD MCORTEX

A. At MDS console, type "SBC861<CR>".

B. IF '*CONTROL*' appears, SBC was not able to set its baud rate. Press RESET on MULTIBUS frame and try again.

C. If 'Bad EMDS connection' appears, you will not be able to continue. Check connectons. Make sure diskette is not write protected. Push RESET at frame. Try again.

D. SBC861 will announce itself and prompt with ".".

E. Type "L KORE<cr>". Wait for ".". At this point the KORE module has been loaded into the SBC memory, and into the shared memory board.

V. SAVING KORE TO CP/M-86 FILE

A. Leaving the SBC861 process active on the MDS system, disconnect the RS232 J2 connector at the SBC, and connect the terminal prepared earlier.

B. At the newly connected terminal type "GFFD4:4<cr>". The CRT will not echo this entry. Respond to the cues that follow as required until CP/M-86 is up.

C. Now enter DDT86. At this point KORE, CP/M-86, and

DDT86 all are resident in the SBC memory and in the 32K shared memory board.

D. Using DDT86 commands, reposition the parts of KORE required so that the code can be saved into one file. Data necessary to determine the initial locations of the code is found in KORE.MP2. The DDT86 instructions used for the current KORE.OPS and KORE.TRC files follows:

*** KORE.OPS ***

MBB0:0,DFF,480:0 *** Move, starting at address BB0:0, DFF bytes of code (main part of KORE) to new start address 480:0.

M439:0,80,560:0 *** Move, starting at address 439:0, 80 bytes of code (initialization module) to new start address 560:0 (following main part as moved above).

ME794:0,6BF,568:0 *** Move, starting at address E794:0, 6BF bytes of code (GLOBAL memory) to new start address 568:0 (following initialization module).

WKORE.OPS,480:0,153F *** Write to the default disk a file called KORE.OPS starting at address 480:0 and containing 153F bytes.

*** KORE.TRC ***

M439:0,25,C68:0 *** Move, starting at address 439:0, 25 bytes of code (initialization module) to new starting address C68:0 (following main KORE code).

MAC0:0,1AFF,439:0 *** Move, starting at address AC0:0, 1AFF bytes of code (main KORE + initialization module) to

new starting address 439:0.

ME794:0,6BF,439:1B00 *** Move, starting at address E794:0, 6BF bytes of code (GLOBAL memory) to new starting address 439:1B00 (following initialiaztion module).

WKORE.TRC,439:0,21C0 *** Write to the default disk a file called KORE.TRC starting at address 439:0 and containing 21C0 bytes.

NOTE: The main KORE module, the initialization module, and GLOBAL memory are located to separate parts of the SBC by the MCORTEX loader. The system used requires that these modules be saved into the file in 128 byte blocks. Further, any change in the number of 128 byte blocks occupied by each must be reflected in the MCORTEX loader code.

APPENDIX B

MCORTEX UNDER DDT86

When troubleshooting MCORTEX processes using DDT86, it is important to realize that DDT86 break points are implemented as 8086 commands written at the locations in memory selected as break points. If 'DDT86 MCORTEX" is executed, the MCORTEX system will be loaded under the control of DDT86. If an attempt is made to execute the loader code to a break point inside a user module which is still to be loaded, DDT86 installs the break point command as directed, but this command will be overwritten when the user code is loaded. The code will execute through the intended break point, and the desired result will not be achieved.

To enable break points within user processes, execute 'DDT86 MCORTEX' as before. Now set a break point inside the MCORTEX loader code, but after KORE and the user processes have been loaded. The loader will now input KORE and user modules as directed, and DDT86 will break inside the loader. At this point further break points within KORE and user code can be successfully set, and will not be overwritten.

Trying to use DDT86 on PL/I-86 code can be very confusing as the 8086 code produced is not familiar. Use the MAP function of the LINK86 linker to give yourself

landmark addresses as you traverse the code. The MAP file gives you beginning addresses for each of your procedures and each of the runtime modules provided by PL/I-86. Similar information is found in the MP2 files for KORE code.

When tracing code, use a hierarchical search. Use go instructions with break points, or individual trace instructions to execute small sections of code at a time. Break points should be set just past the next call to be executed. When a failure occurs, you will have bracketed the possible code causing the error. If the error is within the call, simply trace into the call one trace step, list the code and proceed in the hierarchical manner used before. Note that you must be mindful of jump instructions in the execution path. You may have to trace several bytes of code to ensure that the execution path includes the break address. This procedure will get you to the errant code with the least amount of tracing.

APPENDIX C

MCORTEX LOADER

This file when assembled produces the MCORTEX loader. The loader when invoked from CP/M-86, gives an indication that it is on line, and then asks if GLOBAL memory is to be loaded. The first CPU entering the MCORTEX environment should load GLOBAL memory, all others should not. The last process loaded on each SBC must contain the initialization routine containing all create process functions. This file contains code that is conditionally assembled to create MXTRACE. The value of MCORTEX in the code controls which module is produced, and the name of the file produced must be changed by the user.

```
;*****************************************************************/
;* MCORTEX / MXTRACE   File TEX/TRC.A86   Rowe 13 Feb 84 */
;*---------------------------------------------------------------*/
;* This program loads the MCORTEX operating system from  */
;* disk into the current CP/M environment.  The system   */
;* memory space is reserved using CP/M memory management */
;* functions.  Since INITIALPROC must be over written by */
;* the user INITIALPROC, the memory it occupies is not   */
;* reserved.  The portions loaded into the interrupt     */
;* area and into shared memory (ie. GLOBALMODULE) are in */
;* areas not managed by CP/M and are thus protected from */
;* user overwrite when using PLI CMD files.  Conditional */
;* assemblies allow assembly of either MCORTEX or MXTRACE*/
;* depending on the value assigned to MCORTEX at the     */
;* beginning of the code.  Nine such conditional         */
;* assembly statements are included.                     */
;*****************************************************************/

            DSEG
            ORG  0000H
;*** MCORTEX / MXTRACE SELECTION ***************************/

MCORTEX                     EQU 1 ;*** SET TO ZERO FOR
                                  ;*** MXTRACE, TO ONE FOR
                                  ;*** MCORTEX

;*** ADDRESS CONSTANTS *************************************/

FCB                         EQU 005CH       ;*** FILE CONTROL
FCB_NAME                    EQU 005DH       ;*** BLOCK
FCB_EXTENT                  EQU 0068H
FCB_CR                      EQU 007CH

INT_ADD_CS                  EQU 0011H     ;*** INTERRUPT CODE
INTRPT_OFFSET               EQU 0033H     ;*** SEGMENT AND
IF MCORTEX
INTRPT_CS                   EQU 0C6BH     ;*** VECTOR
ELSE
INTRPT_CS                   EQU 0C31H     ;#### 1 #### <----
ENDIF

;*** PURE NUMBER CONSTANTS *********************************/

EIGHTH_K                    EQU 0080H
IF MCORTEX
NUM_KORE_BLOCKS             EQU 001CH
ELSE
NUM_KORE_BLOCKS             EQU 0035H     ;#### 2 #### <----
ENDIF

ASCII_0                     EQU '0'
ASCII_9                     EQU '9'
```

```
ASCII_A                         EQU 'A'
ASCII_Z                         EQU 'Z'
COLON                           EQU ':'
SPACE                           EQU ' '
PERIOD                          EQU '.'
CR                              EQU 000DH
LF                              EQU 000AH


;*** CONTROL TRANSFER CONSTANTS ***************************/

IF MCORTEX
KORE_SP                         EQU 0080H
KORE_SS_VAL                     EQU 0C78H
KORE_DS_VAL                     EQU 0C69H
ELSE
KORE_SP                         EQU 00FFH   ;#### 3 #### <-----
KORE_SS_VAL                     EQU 0C30H   ;#### 4 #### <-----
KORE_DS_VAL                     EQU 0C00H   ;#### 5 #### <-----
ENDIF

;*** CP/M FUNCTION CONSTANTS ******************************/

CPM_BDOS_CALL                   EQU 224
SYSTEM_RESET                    EQU 0000H
CONSOLE_OUTPUT                  EQU 0002H
READ                            EQU 000AH
PRINT_STRING                    EQU 0009H
OPEN_FILE                       EQU 000FH
READ_SEQUENTIAL                 EQU 0014H
SET_DMA_OFFSET                  EQU 001AH
SET_DMA_BASE                    EQU 0033H
ALLOC_MEM_ABS                   EQU 0038H
FREE_ALL_MEM                    EQU 003AH
PROGRAM_LOAD                    EQU 003BH
NOT_FOUND                       EQU 00FFH

;*** MESSAGES *********************************************/

IN_STRING                       DB 15
                                RB 16

NO_FILE_MSG DB 'KORE NOT ON DEFAULT DRIVE$'
NO_IN_FILE_MSG DB 'INPUT FILE NOT ON DESIGNATED DRIVE$'
NO_MEMORY_MSG DB 'UNABLE TO ALLOCATE MEMORY SPACE FOR'
              DB ' MCORTEX$'
FILE_FORM_ERR_MSG DB 'INCORRECT FILE FORMAT - TRY AGAIN$'

START_MSG DB 'MCORTEX SYSTEM LOADER *** ON LINE$'
P_NAME_MSG DB CR,LF,LF,'ENTER PROCESSOR FILE NAME:',CR,LF
           DB '$'
```

```
GLOBAL_Q_MSG DB CR,LF,LF,'LOAD GLOBAL MEMORY?',CR,LF
GM2_MSG DB '"Y" TO LOAD, "RETURN" TO SKIP',CR,LF,'$'

;*** MCORTEX RELOCATION VARIABLES **************************/

;*** CAUTION *** CAUTION *** CAUTION *** CAUTION **********/
;*** The following five lines of code should not be     ***/
;*** separated as this program assumes they will be     ***/
;*** found in the order shown.  The code is used for    ***/
;*** memory allocation and as a pointer to KORE.        ***/
;*** CAUTION *** CAUTION *** CAUTION *** CAUTION **********/

KORE_START                  DW  0030H           ;*** CAUTION
IF MCORTEX
KORE1_BASE                  DW 0BB0H            ;*** CAUTION
ELSE
KORE1_BASE                  DW 0AC0H    ;#### 6 #### <-----
ENDIF
KORE                    EQU DWORD PTR KORE_START ;*** CAUTION
IF MCORTEX
KORE1_LENGTH                DW 00E0H            ;*** CAUTION
ELSE
KORE1_LENGTH                DW 01C0H    ;#### 7 #### <-----
ENDIF
KORE1_M_EXT                 DB 0                ;*** CAUTION

IF MCORTEX
KORE_NAME                   DB 'KORE    OPS'
ELSE
KORE_NAME                   DB 'KORE    TRC' ;### 8 ### <--
ENDIF

KORE2_BASE                  DW 0E794H ;*** GLOBAL MEMORY

INTERRUPT_VECTOR            DW INTRPT_OFFSET,INTRPT_CS
INT_VECTOR_ADD              DW INT_ADD_CS

INIT_OFFSET         DW 0000H  ;*** INITIALIZATION
INIT_BASE           DW 0439H  ;*** ROUTINE PARAMETERS
IF MCORTEX
INIT_DS_SEG         DW 0C88H  ;*** FOR DYNAMIC ASSIGNMENT
ELSE
INIT_DS_SEG         DW 0C58H        ;#### 9 #### <--------
ENDIF
INIT_DS_OFFSET      DW 0068H  ;*** WHEN USER INITIALIZATION
INIT_IP_OFFSET      DW 0074H  ;*** IS INDICATED

;*** CONTROL TRANSFER VARIABLES ***************************/

KORE_SS                     DW KORE_SS_VAL
KORE_DS                     DW KORE_DS_VAL
```

```
;*** START CODE SEGMENT ********************************/

MCORTEX_LOADER CSEG

CALL CLR_SCREEN         ;*** SCREEN CONTROL & LOG ON
CALL MCORTEX_LOAD       ;*** MESSAGES
CALL CLR_SCREEN         ;***

CLD                     ;*** INITIALIZATION
PUSH AX                 ;***

;*** GET LOAD GLOBAL INDICATOR *************************/

CALL IN_GLOBAL          ;*** ASK IF GLOBAL TO BE LOADED
MOV DX,OFFSET IN_STRING ;*** GET BUFFER LOCATION
MOV CL,READ             ;*** CP/M PARAMETER
INT CPM_BDOS_CALL       ;*** GET INDICATER

;*** GENERATE KORE FILE CONTROL BLOCK ******************/

GEN_KORE_FCB:
MOV BX,10               ;*** MOVE 11 CHARACTERS
MOV SI,OFFSET KORE_NAME ;*** POINT TO KORE NAME
MOV DI,FCB_NAME         ;*** POINT TO FCB NAME
MOV_KORE:
MOV AL,[SI+BX]          ;*** GET CHARACTER
MOV [DI+BX],AL          ;*** STORE CHARACTER
DEC BX
JGE MOV_KORE

;*** OPEN KORE.OPS FILE ON DEFAULT DISK ****************/

OPEN_KORE:
MOV CL, OPEN_FILE               ;*** CP/M PARAMETER
MOV DX,FCB                      ;*** CP/M PARAMETER
INT CPM_BDOS_CALL               ;*** OPEN FILE
CMP AL,NOT_FOUND                ;*** FILE FOUND?
JNE PROCESS_KORE                ;*** FILE FOUND! CONTINUE
JMP NO_FILE                     ;*** GO INDICATE ERROR
PROCESS_KORE:
MOV DI,0
MOV FCB_CR[DI],DI               ;*** START WITH REC ZERO

;*** RESERVE MEMORY ************************************/

MOV CL,FREE_ALL_MEM         ;*** CP/M PARAMETER
INT CPM_BDOS_CALL           ;*** FREE ALL MEMORY
MOV CL,ALLOC_MEM_ABS        ;*** CP/M PARAMETER
MOV DX,OFFSET KORE1_BASE    ;*** CP/M PARAMETER
INT CPM_BDOS_CALL           ;*** ALLOCATE MEMORY
CMP AL, NOT_FOUND           ;*** MEMORY AVAILABLE?
JNE LOAD_MCORTEX            ;*** MEMORY AVAILABLE! CONTINUE
```

```
JMP NO_MEMORY_ALLOC          ;*** GO INDICATE ERROR

;*** LOAD MCORTEX CODE AT ØAC0H **************************/

LOAD_MCORTEX:
MOV DI,Ø                     ;*** SET DEST. OFFSET
MOV BP,NUM_KORE_BLOCKS       ;*** SET BLOCK COUNTER
MOVE_KORE_LOOP:
MOV DX,FCB                   ;*** CP/M PARAMETER
MOV CL,READ_SEQUENTIAL       ;*** CP/M PARAMETER
INT CPM_BDOS_CALL            ;*** READ IN 128 BYTES
MOV ES,KORE1_BASE            ;*** SET DESTINATION SEGMENT
MOV CX,EIGHTH_K              ;*** SET BYTE COUNT
MOV SI,CX                    ;*** SET SOURCE OFFSET
REP  MOVSB                   ;*** MOVE 128 BYTES
DEC BP                       ;*** DEC BLOCKS TO MOVE
JNZ MOVE_KORE_LOOP           ;*** IF NOT DONE, DO AGAIN

;*** LOAD INITIALIZATION MODULE **************************/

MOV DI,INIT_OFFSET           ;*** SET DEST. OFFSET
MOV DX,FCB                   ;*** CP/M PARAMETER
MOV CL,READ_SEQUENTIAL       ;*** CP/M PARAMETER
INT CPM_BDOS_CALL            ;*** READ IN 128 BYTES
MOV ES,INIT_BASE             ;*** SET DESTINATION SEGMENT
MOV CX,EIGHTH_K              ;*** SET BYTE COUNT
MOV SI,CX                    ;*** SET SOURCE OFFSET
REP MOVSB                    ;*** MOVE 128 BYTES

;*** LOAD GLOBAL MEMORY **********************************/

CMP IN_STRING+1,ØH           ;*** SHOULD GLOBAL BE LOADED?
JZ INSTALL_INTERRUPT         ;*** IF NOT, SKIP LOAD
MOV DI,Ø                     ;*** SET DEST. OFFSET
MOVE_GLOBAL_LOOP:
MOV DX,FCB                   ;*** CP/M PARAMETER
MOV CL,READ_SEQUENTIAL       ;*** CP/M PARAMETER
INT CPM_BDOS_CALL            ;*** READ 128 BYTES
TEST AL,AL                   ;*** NO MORE DATA?
JNZ INSTALL_INTERRUPT        ;*** IF NO MORE, GO ON
MOV ES,KORE2_BASE            ;*** SET DESTINATION SEGMENT
MOV CX,EIGHTH_K              ;*** SET BYTE COUNT
MOV SI,CX                    ;*** SET SRC. OFFSET
REP  MOVSB                   ;*** MOVE 128 BYTES
JMP MOVE_GLOBAL_LOOP         ;*** IF NOT DONE, DO AGAIN

;*** INITIALIZE INTERRUPT VECTOR *************************/

INSTALL_INTERRUPT:
MOV ES,INT_VECTOR_ADD        ;*** SET DESTINATION SEGMENT
MOV DI,Ø                     ;*** SET DEST. OFFSET
MOV SI,OFFSET INTERRUPT_VECTOR ;*** SRC. OFFSET
```

```
MOV CX,2                        ;*** 2 WORDS TO MOVE
REP  MOVS AX,AX                 ;*** MOVE TWO WORDS

;*** READ IN A FILE NAME ********************************/

READ_A_NAME:
CALL PROCESSOR_NAME             ;*** MSG TO INPUT A FILE NAME
MOV DX,OFFSET IN_STRING         ;*** DX <-- BUFFER LOCATION
MOV CL,READ                     ;*** CPM PARAMETER
INT CPM_BDOS_CALL               ;*** GET A FILE NAME

;*** SET FCB DRIVE DESIGNATION **************************/

CMP IN_STRING+1,0   ;*** ARE THERE MORE INPUTS?
JE EXIT_ROUTINE_B   ;*** IF NO, GET GLOBAL LOAD INDICATER
POP AX              ;*** LAST LOADED FILE WAS NOT INITIALIZE

MOV DI,0            ;*** SET DESTINATION INDEX TO ZERO

CMP IN_STRING+3,COLON ;*** IS DRIVE DESIGNATED?
JE SET_DRIVE        ;*** IF YES, PUT DRIVE IN FCB
MOV FCB[DI],DI      ;*** SET DEFAULT DRIVE
MOV SI,2            ;*** 3RD POSIT IN_STRING, IS 1ST LETTER
JS FORM_FCB

SET_DRIVE:
MOV AL,IN_STRING+2 ;*** GET DRIVE LETTER
AND AL,5FH         ;*** CONVERT TO UPPER CASE
SUB AL,40H         ;*** CONVERT TO A BINARY NUMBER
MOV FCB[DI],AL     ;*** SET DRIVE
AND AL,0F0H        ;*** LIMIT LINE DRIVE TO A THROUGH O
TEST AL,AL
JNZ INPUT_ERROR_B
MOV SI,4           ;*** 5TH POSIT IN_STRING IS 1ST LETTER

;*** INITIALIZE FILE CONTROL BLOCK **********************/

FORM_FCB:
MOV BX,0AH              ;*** FILL FCB NAME WITH SPACES
MOV AL,SPACE            ;***
FILL_SPACES:
MOV FCB_NAME[BX],AL     ;***
DEC BX                  ;***
JGE FILL_SPACES         ;***

MOV FCB_CR[DI],DI       ;*** NEW FILE CURRENT RECORD IS ZERO
MOV FCB_EXTENT[DI],DI   ;*** NEW FILE CURRENT EXTENT IS ZERO

;*** INSTALL FILE CONTROL BLOCK NAME ********************/

NAME_LOOP:
MOV AL,IN_STRING[SI]   ;*** GET A CHARACTER
```

```
CMP AL,PERIOD           ;*** START TYPE?
JNE FCB_CONT_1          ;*** IF NO, CONTINUE
MOV DI,8                ;*** IF YES, ADJUST DESTINATION
JMP FCB_CONT_2          ;*** AND CONTINUE
FCB_CONT_1:
CALL VALID_INPUT        ;*** CHECK FOR LETTER OR NUMBER
TEST AX,AX              ;***
JE INPUT_ERROR_B        ;***
MOV FCB_NAME[DI],AL     ;*** MOVE CHARACTER INTO FCB
MOV AX,SI               ;*** IS THIS LAST CHARACTER?
CMP IN_STRING+1,AL      ;***
JB OPEN_PROCESSOR       ;*** IF YES, LOAD THE FILE
INC DI                  ;*** IF NO, ADJUST FOR NEXT LETTER
FCB_CONT_2:
INC SI                  ;*** AND GO AGAIN
JMP NAME_LOOP           ;***

EXIT_ROUTINE_B:
JMP EXIT_ROUTINE        ;*** BRIDGE TO EXIT ROUTINE
INPUT_ERROR_B:
JMPF INPUT_ERROR        ;*** BRIDGE TO INPUT_ERROR



;*** OPEN THE PROCESSOR FILE **************************************/

OPEN_PROCESSOR:
MOV DX,FCB              ;*** CP/M PARAMETER
MOV CL,OPEN_FILE        ;*** CP/M PARAMETER
INT CPM_BDOS_CALL       ;*** OPEN THE FILE
CMP AL, NOT_FOUND       ;*** WAS FILE ON DISK
JNE LOAD_PROCESSOR      ;*** IF YES, GO LOAD THE FILE
JMP NO_INPUT_FILE       ;*** IF NO, SIGNAL ERROR
LOAD_PROCESSOR:
MOV DX,FCB              ;*** CP/M PARAMETER
MOV CL,PROGRAM_LOAD     ;*** CP/M PARAMETER
INT CPM_BDOS_CALL       ;*** LOAD THE FILE
PUSH AX                 ;*** SAVE DATA SEGMENT
JMP READ_A_NAME         ;*** GET NEXT PROCESSOR

;*** SET UP THE INITIALIZATION STACK ******************************/

;*** CAUTION *** CAUTION *** CAUTION *** CAUTION **********/
;*** This code is highly dependent upon Input of PL/I   ***/
;*** CMD file with CS header first and data header      ***/
;*** second.  This is the normal situation and should   ***/
;*** cause no difficulty.  Also this code is highly     ***/
;*** dependent upon the location of the initialization  ***/
;*** module stack and the location of the DS and IP     ***/
;*** values within that stack.  Changes in stack        ***/
;*** location or organization should be reflected here.***/
```

```
;*** CAUTION *** CAUTION *** CAUTION *** CAUTION *********/

EXIT_ROUTINE:
POP AX                          ;*** RECOVER DATA SEGMENT
MOV ES,INIT_DS_SEG              ;*** POINT TO INIT STACK
MOV BX,INIT_DS_OFFSET           ;*** POINT TO DS ON STACK
MOV ES:[BX],AX                  ;*** INSTALL NEW INIT DS
MOV DX,0                        ;*** SET NEW IP VALUE
MOV BX,INIT_IP_OFFSET           ;*** POINT TO IP ON STACK
MOV ES:[BX],DX                  ;*** INSTALL NEW INIT IP
MOV CL,SET_DMA_BASE             ;*** CP/M PARAMETER
MOV DX,AX                       ;*** SET BASE PAGE
INT CPM_BDOS_CALL               ;*** SET DMA BASE
MOV CL,SET_DMA_OFFSET           ;*** CP/M PARAMETER
MOV DX,EIGHTH_K                 ;*** GET OFFSET
INT CPM_BDOS_CALL               ;*** SET DMA OFFSET

;*** TRANSFER CONTROL TO MCORTEX ****************************/

MOV SP,KORE_SP                      ;*** KORE STACK POINTER
MOV BP,SP                           ;*** KORE STACK BASE
MOV SS,KORE_SS                      ;*** KOIE STACK SEGMENT
MOV AX,DS                           ;*** GET DATA SEGMENT
MOV ES,AX                           ;*** POINT ES TO DS
MOV DS,KORE_DS                      ;*** KORE DATA SEGMENT
JMPF ES:KORE                        ;*** JUMP TO MCORTEX

;*** VALID CHARACTER FOR FILE NAME CHECK ****************/

VALID_INPUT:
CMP AL,ASCII_0          ;*** IS THE CHARACTER A NUMBER
JB NOT_VALID            ;***
CMP AL,ASCII_9          ;***
JBE IS_VALID            ;***
AND AL,5FH              ;*** CONVERT CHARACTER TO UPPER CASE
CMP AL,ASCII_A          ;*** IS THE CHARACTER A LETTER
JB NOT_VALID            ;***
CMP AL,ASCII_Z          ;***
JBE IS_VALID            ;***
NOT_VALID:
MOV AX,0                ;*** INDICATE BAD CHARACTER
IS_VALID:
RET                     ;*** CHARACTER OK

;*** ABORT MESSAGES *************************************/

NO_FILE:
CALL CLR_SCREEN
MOV DX,OFFSET NO_FILE_MSG     ;*** PTR TO MSG
JMP MSG_OUTPUT                ;*** PUT MSG

NO_MEMORY_ALLOC:
```

```
CALL CLR_SCREEN
MOV DX,OFFSET NO_MEMORY_MSG  ;*** PTR TO MSG

MSG_OUTPUT:
MOV CL,PRINT_STRING          ;*** CP/M PARAMETER
INT CPM_BDOS_CALL            ;*** SEND CHAR TO CONSOLE
CALL CLR_SCREEN
MOV CL,SYSTEM_RESET          ;*** CP/M PARAMETER
MOV DL,0                     ;*** RELEASE MEMORY
INT CPM_BDOS_CALL            ;*** EXIT TO CP/M

;*** SCREEN CONTROL ***************************************/

CLR_SCREEN:
MOV CL,CONSOLE_OUTPUT    ;*** ISSUE CARRIAGE RETURN
MOV DL,CR                ;***
INT CPM_BDOS_CALL        ;***
MOV DI,0CH               ;*** ISSUE 12 LINE FEEDS
LINE_FEED:
MOV DL,LF                ;***
MOV CL,CONSOLE_OUTPUT    ;***
INT CPM_BDOS_CALL        ;***
DEC DI                   ;***
JNE LINE_FEED            ;***
RET

SEND_MSG:
MOV CL,PRINT_STRING      ;*** CP/M PARAMETER
INT CPM_BDOS_CALL        ;*** PRINT A STRING TO CONSOLE
RET

;*** NON ABORT MESSAGES ***********************************/

MCORTEX_LOAD:
MOV DX,OFFSET START_MSG
CALL SEND_MSG
RET

PROCESSOR_NAME:
MOV DX,OFFSET P_NAME_MSG
CALL SEND_MSG
RET

IN_GLOBAL:
MOV DX,OFFSET GLOBAL_Q_MSG
CALL SEND_MSG
RET

INPUT_ERROR:
CALL CLR_SCREEN
MOV DX,OFFSET FILE_FORM_ERR_MSG
JMP EXIT_ERR
```

```
NO_INPUT_FILE:
CALL CLR_SCREEN
MOV DX,OFFSET NO_IN_FILE_MSG
EXIT_ERR:
CALL SEND_MSG
CALL CLR_SCREEN
JMP READ_A_NAME

END
```

APPENDIX D

GATE MODULE CODE

Two files are contained here. The first is PL/I code, GATEWAY, which must be %INCLUDE'd with every user process requiring access to MCORTEX. The second is A86 code which provides an interface between the GATEWAY and the MCORTEX supervisor. The object code obtained from assembly of this file must be linked with all user processes to provide "gateway" access to MCORTEX functions. Two lines of code are conditionally assembled to produce either GATEMOD or GATETRACE. The conditional variable is called GATEMOD.

```
/*****************************************************************/
/*****************************************************************/
/** GATEWAY     FILE GATEWAY.PLI     W.R. ROWE  4 MAR 84 **/
/**=============================================================**/
/** This section of code is given as a PLI file to be     **/
/** %INCLUDE'd with MCORTEX user programs.  ENTRY         **/
/** declarations are made for all available MCORTEX       **/
/** functions and for ADD2BIT16, a utility function       **/
/** allowing unsigned addition of 16 bit numbers.         **/
/*****************************************************************/
/*****************************************************************/

    DECLARE

      advance ENTRY (BIT (8)),
      /* advance (event_count_id) */

      await ENTRY (BIT (8), BIT (16)),
      /* await (event_count_id, awaited_value) */

      create_evc ENTRY (BIT (8)),
      /* create_evc (event_count_id) */

      create_proc ENTRY (BIT (8), BIT (8),
                         BIT (16), BIT (16), BIT (16),
                         BIT (16), BIT (16), BIT (16)),
     /* create_proc (processor_id, processor_priority,    */
     /*              stack_pointer_highest, stack_seg, ip */
     /*              code_seg, data_seg, extra_seg)       */

      create_seq ENTRY (BIT (8)),
        /* create_seq (sequence_id) */

      preempt ENTRY (BIT (8)),
        /* preempt (processor_id) */

      read ENTRY (BIT (8)) RETURNS (BIT (16)),
        /* read (event_count_id) */
        /* RETURNS current_event_count */

      ticket ENTRY (BIT (8)) RETURNS (BIT (16)),
        /* ticket (sequence_id) */
        /* RETURNS unique_ticket_value */

      add2bit16 ENTRY (BIT(16), BIT(16)) RETURNS (BIT (16));
        /* add2bit16 ( a_16bit_#, another_16bit_#) */
        /* RETURNS a_16bit_# + another_16bit_#     */
```

```
;*****************************************************************/
;* GATEMOD / GATETRC   File GATEM/T.a86    Rowe 12 Feb 84 */
;*---------------------------------------------------------------*/
;* This module is given to the user in obj form to link   */
;* with his initial and process modules.  Any changes to  */
;* user services available from the OS must be reflected  */
;* here.  In this way the user need not be concerned with */
;* actual GATEKEEPER services codes.  Two lines of code   */
;* are contained in conditional assembly statements and   */
;* control the output to be GATEMOD or GATETRC depending  */
;* on the value of GATEMOD at the code start.             */
;*---------------------------------------------------------------*/
;* This module reconciles parameter passing anomalies     */
;* between MCORTEX (written in PL/M) and user programs    */
;* (written in PL/I).                                     */
;*---------------------------------------------------------------*/
;* All calls are made to the GATEKEEPER in LEVEL2 of the  */
;* OS.  The address of the GATEKEEPER must be given below.*/
;*---------------------------------------------------------------*/
;* The ADD2BIT16 function does not make calls to MCORTEX. */
;* It's purpose is to allow the addition of two unsigned  */
;* 16 bit numbers from PL/I programs.                     */
;*****************************************************************/

DSEG

GATEMOD EQU 0 ;*** SET TO ZERO FOR GATETRC
              ;*** SET TO ONE FOR GATEMOD

PUBLIC ADVANCE       ;*** THESE DECLARATIONS MAKE THE
PUBLIC AWAIT         ;*** GATEKEEPER FUNCTIONS VISIBLE
PUBLIC CREATE_EVC    ;*** TO EXTERNAL PROCESSES
PUBLIC CREATE_PROC
PUBLIC CREATE_SEQ
PUBLIC PREEMPT
PUBLIC READ
PUBLIC TICKET
PUBLIC ADD2BIT16

AWAIT_IND EQU 0          ;*** THESE ARE THE IDENTIFICATION
ADVANCE_IND EQU 1        ;*** CODES RECOGNIZED BY THE
CREATE_EVC_IND EQU 2     ;*** GATEKEEPER IN LEVEL II OF
CREATE_SEQ_IND EQU 3     ;*** MCORTEX
TICKET_IND EQU 4
READ_IND EQU 5
CREATE_PROC_IND EQU 6
PREEMPT_IND EQU 7

IF GATEMOD
GATEKEEPER_IP DW 002AH
GATEKEEPER_CS DW 0BEBH
ELSE
```

```
GATEKEEPER_IP DW 0062H          ;#### 1 #### <----------------
GATEKEEPER_CS DW 0B4AH          ;#### 2 #### <----------------
ENDIF
GATEKEEPER ECU DWORD PTR GATEKEEPER_IP

CSEG

;*** AWAIT *** AWAIT *** AWAIT *** AWAIT *** AWAIT *********/

AWAIT:
PUSH ES
MOV SI,2[BX]            ;SI <-- PNT TO COUNT AWAITED
MOV BX,[BX]             ;BX <-- PNT TO NAME OF EVENT
MOV AL,AWAIT_IND
PUSH AX                 ;N <-- AWAIT INDICATOR
MOV AL,[BX]
PUSH AX                 ;BYT <-- NAME OF EVENT
MOV AX,[SI]             ;AX <-- COUNT AWAITED
PUSH AX                 ;WORDS <-- COUNT AWAITED
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES
RET

;*** ADVANCE *** ADVANCE *** ADVANCE *** ADVANCE ***********/

ADVANCE:
PUSH ES
MOV BX,[BX]             ;BX <-- PTR TO NAME OF EVENT
MOV AL,ADVANCE_IND
PUSH AX                 ;N <-- ADVANCE INDICATER
MOV AL,[BX]
PUSH AX                 ;BYT <-- NAME OF EVENT
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES
RET

;*** CREATE_EVC *** CREATE_EVC *** CREATE_EVC *************/

CREATE_EVC:
PUSH ES
MOV BX,[BX]             ;BX <-- PTR TO NAME OF EVENT
MOV AL,CREATE_EVC_IND
PUSH AX                 ;N <-- CREATE_EVC INDICATOR
MOV AL,[BX]
PUSH AX                 ;BYT <-- NAME OF EVENT
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
```

```
PUSH AX                 ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES
RET

;*** CREATE_SEQ *** CREATE_SEQ *** CREATE_SEQ ************/

CREATE_SEQ:
PUSH ES
MOV BX,[BX]             ;BX <-- PTR TO NAME OF SEQ
MOV AL,CREATE_SEQ_IND
PUSH AX                 ;N <-- CREATE_SEQ INDICATER
MOV AL,[BX]
PUSH AX                 ;BYT <-- NAME OF SEQ
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES
RET

;*** TICKET *** TICKET *** TICKET *** TICKET *** TICKET ***/

TICKET:
PUSH ES
PUSH ES                 ;TICKET NUMBER DUMMY STORAGE
MOV CX,SP               ;POINTER TO TICKET NUMBER
MOV BX,[BX]             ;BX <-- PTR TO TICKET NAME
MOV AL,TICKET_IND
PUSH AX                 ;N <-- TICKET INDICATER
MOV AL,[BX]
PUSH AX                 ;BYT <-- TICKET NAME
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH SS                 ;PTR_SEG <-- TICKET NUMBER SEG
PUSH CX                 ;PTR_OFFSET <-- TICKET NUMBER POINTER
CALLF GATEKEEPER
POP BX                  ;RETRIEVE TICKET NUMBER
POP ES
RET

;*** READ *** READ *** READ *** READ *** READ *** READ ****/

READ:
PUSH ES
PUSH ES                 ;EVENT COUNT DUMMY STORAGE
MOV CX,SP               ;POINTER TO EVENT COUNT
MOV BX,[BX]             ;BX <-- PTR TO EVENT NAME
MOV AL,READ_IND
PUSH AX                 ;N <-- READ INDICATER
MOV AL,[BX]
PUSH AX                 ;BYT <-- EVENT NAME
PUSH AX                 ;BYT <-- UNUSED WORD
```

```
PUSH SS                 ;PTR_SEG <-- EVENT COUNT SEGMENT
PUSH CX                 ;PTR_OFFSET <-- EVENT COUNT POINTER
CALLF GATEKEEPER
POP BX                  ;RETRIEVE EVENT COUNT
POP ES
RET

;*** CREATE_PROC *** CREATE_PROC *** CREATE_PROC **********/

CREATE_PROC:
PUSH ES
MOV SI,14[BX]           ;SI <-- PTR TO PROCESS ES
PUSH WORD PTR [SI]      ;STACK PROCESS ES
MOV SI,12[BX]           ;SI <-- PTR TO PROCESS DS
PUSH WORD PTR [SI]      ;STACK PROCESS DS
MOV SI, 10[BX]          ;SI <-- PTR TO PROCESS CS
PUSH WORD PTR [SI]      ;STACK PROCESS CS
MOV SI, 8[BX]           ;SI <-- PTR TO PROCESS IP
PUSH WORD PTR [SI]      ;STACK PROCESS IP
MOV SI, 6[BX]           ;SI <-- PTR TO PROCESS SS
PUSH WORD PTR [SI]      ;STACK PROCESS SS
MOV SI, 4[BX]           ;SI <-- PTR TO PROCESS SP
PUSH WORD PTR [SI]      ;STACK PROCESS SP
MOV SI,2[BX]            ;SI <-- PTR TO PROCESS PRIORITY
MOV AH,[SI]             ;GET PROCESS PRIORITY
MOV SI,[BX]             ;SI <-- PTR TO PROCESS ID
MOV AL,[SI]             ;GET PROCESS ID
PUSH AX                 ;STACK PROCESS PRIORITY AND ID
MOV CX,SP               ;POINTER TO DATA
MOV AL,CREATE_PROC_IND
PUSH AX                 ;N <-- CREATE PROCESS IND
PUSH AX                 ;BYT <-- UNUSED WORD
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH SS                 ;PROC_PTR SEGMENT <-- STACK SEG
PUSH CX                 ;PROC_PTR OFFSET <-- DATA POINTER
CALLF GATEKEEPER
ADD SP,14               ;REMOVE STACKED DATA
POP ES
RET

;*** PREEMPT *** PREEMPT *** PREEMPT *** PREEMPT **********/

PREEMPT:
PUSH ES
MOV BX,[BX]             ;BX <-- PTR TO NAME OF PROCESS
MOV AL,PREEMPT_IND
PUSH AX                 ;N <-- PREEMPT INDICATER
MOV AL,[BX]
PUSH AX                 ;BYTE <-- PREEMPT PROCESS NAME
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <-- UNUSED WORD
```

```
CALLF GATEKEEPER
POP ES
RET

;*** ADD2BIT16 *** ADD2BIT16 *** ADD2BIT16 *** ADD2BIT16 **/

ADD2BIT16:
MOV SI,[BX]             ;SI <-- PTR TO BIT(16)#1
MOV BX,2[BX]            ;BX <-- PTR TO BIT(16)#2
MOV BX,[BX]             ;BX <-- BIT(16)#2
ADD BX,[SI]             ;BX <-- BIT(16)#1 + BIT(16)#2
RET

END
```

APPENDIX E

DEMONSTRATION PROGRAM

The files presented here are a series of procedures that can be compiled separately and linked in accordance with LINK86 input option files in APPENDIX F. The results will be demonstration processes D1 and D2, or D1TRC and D2TRC depending on the option files selected.

```
*****************************************************************
*****************************************************************
***                                                           ***
***                      DINIT.PLI code                       ***
*** This code creates the D1 process for execution under    ***
*** MCORTEX.  Using the MCORTEX loader, the last process    ***
*** to be loaded must contain the initialization process.   ***
***                                                           ***
*****************************************************************
*****************************************************************
init:
  PROCEDURE OPTIONS(MAIN) RETURNS();
    %INCLUDE 'gateway.pli';
    BEGIN;
      CALL create_proc ('01'B4, 'fd'B4,
                        '08db'B4, '070a'B4, '0023'B4,
                        '0439'B4, '070a'B4, '070a'B4);
    /*CALL create_proc (PROCESS_ID, PROCESS_PRIORITY,      */
    /*                  SP          SS          IP         */
    /*                  CS          DS          ES     );  */

      CALL await ('fe'B4, '01'B4);
    /*CALL await (  EVC ,  COUNT); */
    END;
  END;

*****************************************************************
*****************************************************************
***                                                           ***
***                     DINIT2.PLI code                       ***
*** This code creates the D2 process and the delayer        ***
*** process for execution under MCORTEX.  Using the         ***
*** MCORTEX loader the last process loaded must contain     ***
*** the initialization process.                             ***
***                                                           ***
*****************************************************************
*****************************************************************
init:
  PROCEDURE OPTIONS(MAIN) RETURNS();
    %INCLUDE 'gateway.pli';
    BEGIN;
      CALL create_proc ('01'B4, 'ed'B4,
                        '0929'B4, '0713'B4, '0029'B4,
                        '0439'B4, '0713'B4, '0713'B4);

    /*CALL create_proc (PROCESS_ID, PROCESS_PRI            */
    /*                  SP          SS          IP         */
    /*                  CS          DS          ES     );  */
      CALL create_proc ('02'B4, 'fd'B4,
                        '0a49'B4, '0713'B4, '01c2'B4,
                        '0439'B4, '0713'B4, '0713'B4);
```

```
      CALL await ('fe'B4, '01'B4);
    /*CALL await (  EVC ,  COUNT);  */

    END;
  END;

/*****************************************************************
*****************************************************************
***                                                           ***
***                    MCDEMO.PLI code                        ***
*** This code is the main controlling code for the            ***
*** demonstration programs D1 and D2.  It is compiled         ***
*** separately and linked using the D1 and D2 input           ***
*** option files.                                             ***
***                                                           ***
*****************************************************************
*****************************************************************/
mcdemo:
  PROCEDURE;

    %INCLUDE 'gateway.pli';

  DECLARE
    log_on ENTRY;

  DECLARE
    delay_value BIT(16) STATIC INITIAL ('0000'B4),
    one BIT(16) STATIC INITIAL ('0001'B4),
    enough BIT(16) STATIC INITIAL ('0064'B4),

    delay BIT(8) STATIC INITIAL ('02'B4),
    sync BIT(8) STATIC INITIAL ('03'B4),
    exit BIT(8) STATIC INITIAL ('ff'B4);

  DECLARE
    msg1 CHARACTER(21) STATIC INITIAL
      ('Delay Event Count is ');

  CALL log_on;
  CALL create_evc (delay);
  CALL create_evc (sync);
  DO WHILE (delay_value < enough);
    PUT EDIT (msg1, delay_value) (SKIP(5), A(21), B4(4));
    CALL advance (sync);
    CALL await (delay, delay_value);
    delay_value = read (delay);
    delay_value = add2bit16 (delay_value, one);
  END; /* DO WHILE */
  CALL preempt (exit);
END mcdemo;
```

```
*****************************************************************
*****************************************************************
***                                                           ***
***                    LOG_ON.PLI code                        ***
*** This code allows the operater to start all real          ***
*** processors executing in MCDEMO at the same time          ***
*** regardle[ of the order that they came on line.           ***
*** This is a demonstration only and is not required         ***
*** under MCORTEX.                                           ***
***                                                           ***
*****************************************************************
*****************************************************************
log_on:
  PROCEDURE;

    %INCLUDE 'gateway.pli';

  DECLARE
    go_signal CHAR VARYING,
    num_sbc_less_1 BIT(16) STATIC INITIAL ('0001'B4),
    one BIT(16) STATIC INITIAL ('0001'B4),

    turn BIT(16) STATIC INITIAL ('0000'B4),

    log_in BIT(8) STATIC INITIAL ('01'B4);

  DECLARE
    msg1 CHARACTER(39) STATIC INITIAL
      ('MCORTEX Demonstration Program "ON LINE"'),
    msg2 CHARACTER(30) STATIC INITIAL
      ('Press "M" "RETURN" to Continue'),
    msg3 CHARACTER(14) STATIC INITIAL
      ('Turn Value is ');

  PUT EDIT (msg1) (SKIP(12), X(21), A(39));
  PUT EDIT ('') (SKIP(13), A(0));
  CALL create_evc (log_in);
  CALL create_seq (log_in);
  turn = ticket (log_in);
  PUT EDIT (msg3, turn) (A(14), B4(4));
  IF turn = num_sbc_less_1 THEN
    DO;
      PUT EDIT (msg2) (SKIP, X(25), A(30));
      GET LIST (go_signal);
    END; /* DO */
  ELSE
    DO;
      turn = add2bit16 (turn, one);
      PUT EDIT ('ENTER await(log_in, turn) = await(', log_in,
                ', ', turn, ')') (SKIP, A(34), B4(2), A(2),
                B4(4), A(1));
      CALL await (log_in, turn);
```

```
    END; /* DO */
  CALL advance (log_in);
END log_on;

/*****************************************************************
******************************************************************
***                                                            ***
***                    DELAYER.PLI code                        ***
*** This code provides a time delay to demonstration          ***
*** programs D1 and D2, under the control of D2.              ***
***                                                            ***
******************************************************************
******************************************************************/
delayer:
  PROCEDURE;

    %INCLUDE 'gateway.pli';

  DECLARE
    max_count FIXED STATIC INITIAL (16000),
    iterations FIXED STATIC INITIAL (10),
    'k,i,j) FIXED,

    start BIT(16) STATIC INITIAL ('0000'B4),
    num_processors BIT(16) STATIC INITIAL ('0002'B4),

    delay BIT(8) STATIC INITIAL ('02'B4),
    sync BIT(8) STATIC INITIAL ('03'B4);

    DO k = 1 to max_count;
      DO i = 1 to iterations;
        DO j = 1 to max_count;
          END; /* DO */
      END; /* DO */
      CALL advance (delay);
      start = add2bit16 (start, num_processors);
      PUT EDIT ('sync await is ', start) (skip, A(17), B4(4));
      CALL await (sync, start);
    END; /* DO */
END delayer;
```

APPENDIX F

LINK86 INPUT OPTION FILES

This group of files allows linkage of specified object code modules using the LINK86 input abbreviation. As an example, after compilation of DINIT.PLI, MCDEMO.PLI, and LCGON.PLI, and assembly of GATEMOD, the demonstration program D1 is created envoking "LINK86 D1[i]". For further information on input option files, see [Ref. 13].

```
**********************************************************************
**********************************************************************
***                  MCORTEX input option file                     ***
**********************************************************************
**********************************************************************
MCORTEX = TEX/TRC [code[ab[B80]],data[ab[B5C]]]

**********************************************************************
**********************************************************************
***                     D1 input option file                       ***
**********************************************************************
**********************************************************************
D1 =
DINIT [code[ab[54D]], data[ab[439],m[0],ad[82]], map[all]],
MCDEMO,
LOGON,
GATEMOD

**********************************************************************
**********************************************************************
***                     D2 input option file                       ***
**********************************************************************
**********************************************************************
D2 =
DINIT2 [code[ab[54D]], data[ab[439],m[0],ad[82]], map[all]],
MCDEMO,
LOGON,
DELAYER,
GATEMOD

**********************************************************************
**********************************************************************
***                  MXTRACE input option file                     ***
**********************************************************************
**********************************************************************
MXTRACE = TEX/TRC [code[ab[A90]],data[ab[A6C]]]

**********************************************************************
**********************************************************************
***                    D1TRC input option file                     ***
**********************************************************************
**********************************************************************
D1TRC =
DINIT [code[ab[54F]], data[ab[439],m[0],ad[82]], map[all]],
MCDEMO,
LOGON,
GATETRC
```

```
**********************************************************************
**********************************************************************
***                    D2TRC input option file                     ***
**********************************************************************
**********************************************************************
D2TRC =
DINIT2 [code[ab[54F]], data[ab[439],m[0],ad[82]], map[all]],
MCIEMO,
LOGON,
DELAYER,
GATETRC
```

APPENDIX G

LEVEL II -- MCORTEX SOURCE CODE

All the LEVEL II source code written in PL/M is contained in the file LEVEL2.SRC. It is compiled with the LARGE attribute. LEVEL II is one of the relocateable code modules in file: KORE.LNK. It is part of the executable code module in file: KORE. KORE is the development system version of the file KORE.OPS loaded by MCORTEX.CMD under the CP/M-86 operating system. Two memory maps (KORE.OPS and KORE.TRC) located in Appendix H give information on this module. The maps come from file: KORE.MP2 after compiling, linking and locating the applicable files. KORE(OPS) is produced with the code unaltered. KORE(TRC) is obtained by removing and adding appropriate comment marks from the indicated code before processing.

```
/**************************************************************/
/**************************************************************/
/*0009*************************************************************/
/* FILE:          LEVEL2.SRC
   VERSION:       ROWE 6-22-84
   PROCEDURES
       DEFINED:     GATE$KEEPER              CREATE$EVC
                    READ                     AWAIT
                    ADVANCE                  PREEMPT
                    TICKET                   CREATE$PROC
                    OUT$CHAR                 OUT$LINE
                    OUT$NUM                  OUT$DNUM
                    SEND$CHAR                OUT$HEX
                    RECV$CHAR                IN$CHAR
                    IN$NUM                   IN$DNUM
                    IN$HEX

   REMARKS:  !!! CAUTION !!! !!! CAUTION !!! !!! CAUTION!!!
             IF NEW USER SERVICES ARE ADDED TO THIS MODULE
             OR CHANGES ARE MADE TO EXISTING ONES, MAKE
             SURE THE LOCATOR MAP (FILE: KORE.MP2) IS CHECK-
             ED TO SEE IF THE LOCATION OF 'GATE$KEEPER' HAS
             NOT CHANGED.  THE APSOLUTE ADDRESS OF THIS
             PROCEDURE HAS BEEN SUPPLIED TO THE GATE$MODULE
             IN FILE: GATE.SRC.  IF IT HAS CHANGED THE NEW
             ADDRESS SHOULD BE UPDATED IN FILE:  GATE.SRC
             AND RECOMPILED.  ALL USER PROCESSES WILL HAVE
             TO BE RELINKED WITH FILE:  GATE.OBJ AND
             RELOCATED.

             LITERAL DECLARATIONS GIVEN AT THE BEGINNING
             OF SEVERAL MODULES ARE LOCAL TO THE ENTIRE
             MODULE.  HOWEVER, SOME ARE LISTED THE SAME
             IN MORE THAN ONE MODULE.  THE VALUE AND
             THEREFORE THE MEANING OF THE LITERAL IS
             COMMUNICATED ACROSS MODULE BOUNDARIES.
             'NOT$FOUND' USED IN LOCATE$EVC AND
             CREATE$EVC IS AN EXAMPLE.  TO CHANGE IT IN
             ONE MODULE AND NOT THE OTHER WOULD KILL
             THE CREATION OF ANY NEW EVENTCOUNTS BY THE
             OS.

                                                          */
/**************************************************************/
/*0073*************************************************************/

L2$MODULE: DO;


/**************************************************************/
/**************************************************************/
/* LOCAL DECLARATIONS                                         */
```

```
DECLARE
   MAX$CPU                    LITERALLY      '10',
   MAX$VPS$CPU                LITERALLY      '10',
   MAX$CPU$$$$MAX$VPS$CPU     LITERALLY     '100',
   FALSE                      LITERALLY       '0',
   READY                      LITERALLY       '1',
   RUNNING                    LITERALLY       '3',
   WAITING                    LITERALLY       '7',
   TRUE                       LITERALLY     '119',
   NOT$FOUND                  LITERALLY     '255',
   PORT$CA                    LITERALLY   '00CAH',
   RESET                      LITERALLY       '0',
   INT$RETURN                 LITERALLY     '77H';

/*0096**************************************************************/
/* PROCESSOR DATA SEGMENT TABLE                                     */
/*    DECLARED PUBLIC IN MODULE 'L1$MODULE'                         */
/*                    IN FILE   'LEVEL1   '                         */

DECLARE PRDS STRUCTURE
  (CPU$NUMBER                 BYTE,
   VP$START                   BYTE,
   VP$END                     BYTE,
   VPS$PER$CPU                BYTE,
   LAST$RUN                   BYTE,
   COUNTER                    WORD)          EXTERNAL;

/*0109**************************************************************/
/* GLOBAL DATA BASE DECLARATIONS                                    */
/*    DECLARED PUBLIC IN FILE  'GLOBAL.SRC'                         */
/*                    IN MODULE'GLOBAL$MODULE'                      */


DECLARE VPM( MAX$CPU$$$$MAX$VPS$CPU ) STRUCTURE
  (VP$ID                      BYTE,
   STATE                      BYTE,
   VP$PRIORITY                BYTE,
   EVC$THREAD                 BYTE,
   EVC$AW$VALUE               WORD,
   SP$REG                     WORD,
   SS$REG                     WORD)          EXTERNAL;

DECLARE
   EVENTS                     BYTE           EXTERNAL;

DECLARE EVC$TBL (100) STRUCTURE
  (EVC$NAME                   BYTE,
   VALUE                      WORD,
   THREAD                     BYTE)          EXTERNAL;

DECLARE
```

```
   SEQUENCERS                BYTE                EXTERNAL;

DECLARE SEQ$TABLE (100) STRUCTURE
  (SEQ$NAME                  BYTE,
   SEQ$VALUE                 WORD)               EXTERNAL;

DECLARE
   NR$VPS( MAX$CPU )         BYTE                EXTERNAL,
   NR$RPS                    BYTE                EXTERNAL,
   HDW$INT$FLAG (MAX$CPU )BYTE                   EXTERNAL,
   GLOBAL$LOCK               BYTE                EXTERNAL;

/*0166*************************************************************/
/* DECLARATION OF EXTERNAL PROCEDURE REFERENCES                    */
/*    DECLARED PUBLIC IN FILE  'LEVEL1.SRC'                        */
/*                     IN MODULE 'LEVEL1$MODULE'                   */

VPSCHEDULER:  PROCEDURE EXTERNAL; END;
    IN FILE 'SCHED.ASM' */

RET$VP     :   PROCEDURE BYTE EXTERNAL; END;

LOCATE$EVC :   PROCEDURE (EVENT$NAME) BYTE EXTERNAL;
   DECLARE EVENT$NAME BYTE;
END;

LOCATE$SEQ :   PROCEDURE (SEQ$NAME) BYTE EXTERNAL;
   DECLARE SEQ$NAME BYTE;
END;

/*0178*************************************************************/
/* DIAGNOSTIC MESSAGES (WILL EVENTUALLY BE REMOVED)                */

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* DECLARE
/* MSG16(*) BYTE INITIAL('ENTERING  PREEMPT',13,10,'%'),
/* MSG17(*) BYTE INITIAL('ISSUING   INTERRUPT!!',13,10,'%'),
/* MSG18(*) BYTE INITIAL('ENTERING  AWAIT',10,13,'%'),
/* MSG19(*) BYTE INITIAL('ENTERING  ADVANCE ',10,13,'%'),
/* MSG21(*) BYTE INITIAL('ENTERING  CREATE$EVC FOR %'),
/* MSG23(*) BYTE INITIAL('ENTERING  READ FOR EVC:  $'),
/* MSG24(*) BYTE INITIAL('ENTERING  TICKET',13,10,'%'),
/* MSG25(*) BYTE INITIAL('ENTERING  CREATE$SEQ  %'),
/* MSG26(*) BYTE INITIAL('ENTERING  CREATE$PROC',10,13,'%'),
/* MSG27(*) BYTE INITIAL(10,'ENTERING  GATE$KEEPER  N= %');

/* DECLARE
/*    CR LITERALLY  '0DH',
/*    LF LITERALLY  '0AH';

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
```

```
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

/*0218***************************************************************/
/*********************************************************************/
/**  GATE$KEEPER   PROCEDURE                    ROWE 6-22-84 ****/
/*********************************************************************/
/*  THIS PROCEDURE IS THE ENTRY INTO THE OPERATING                   */
/*  SYSTEM DOMAIN FROM THE USER DOMAIN.  THIS IS THE                 */
/*  ACCESS POINT TO THE UTILITY/SERVICE ROUTINES AVAIL-              */
/*  ABLE TO THE USER. THIS PROCEDURE IS CALLED BY THE                */
/*  GATE MODULE WHICH IS LINKED WITH THE USER PROGRAM.               */
/*  IT IS THE GATE MODULE WHICH PROVIDES TRANSLATION                 */
/*  FROM THE USER DESIRED FUNCTION TO THE FORMAT REQUIR-             */
/*  ED FOR THE GATEKEEPER.  THE GATEKEEPER CALLS THE                 */
/*  DESIRED UTILITY/SERVICE PROCEDURE IN LEVEL2 OF THE               */
/*  OPERATING SYSTEM AGAIN PERFORMING THE NECESSARY                  */
/*  TRANSLATION FOR A PROPER CALL.  THE TRANSLATIONS ARE             */
/*  INVISIBLE TO THE USER.  THE GATEKEEPER ADDRESS IS                */
/*  PROVIDED TO THE GATE MODULE TO BE USED FOR THE IN-               */
/*  DIRECT CALL.                                                     */
/*                                                                   */
/*  THE PARAMETER LIST IS PROVIDED FOR CONVENIENCE AND               */
/*  REPRESENTS NO FIXED MEANING, EXCEPT FOR 'N'.                     */
/*     N       FUNCTION CODE PROVIDED BY GATE                        */
/*     BYT     BYTE VARIABLE FOR TRANSLATION                         */
/*     WORDS   WORD                                                  */
/*     PTR     POINTER VARIABLE FOR TRANSLATION                      */
/*0243***************************************************************/

GATE$KEEPER: PROCEDURE(N, BYT, WORDS, PTR) REENTRANT PUBLIC;

   DECLARE
     (N, BYT) BYTE,
      WORDS WORD,
      PTR POINTER;
/* I-O SERVICES ARE NOT ACKNOWLEDGED FOR TWO REASONS:                */
/*    1.  THEY ARE CALLED SO OFTEN THAT DIAGNOSTIC OUTPUT            */
/*        WOULD BE TOO CLUTTERED.                                    */
/*    2.  THEY THEMSELVES PRODUCES I-O EFFECTS THAT                  */
/*        ACKNOWLEDGE THEY ARE BEING CALLED.                         */

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* IF N < 8 THEN DO;
/*    CALL OUT$LINE(@MSG27);
/*    CALL OUT$NUM(N);
/*    CALL OUT$CHAR(CR);
/*    CALL OUT$CHAR(LF);
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
```

```
   DO CASE N;                          /*  N  */
      CALL AWAIT(BYT,WORDS);           /*  0  */
      CALL ADVANCE(BYT);               /*  1  */
      CALL CREATE$EVC(BYT);            /*  2  */
      CALL CREATE$SEQ(BYT);            /*  3  */
      CALL TICKET(BYT,PTR);            /*  4  */
      CALL READ(BYT,PTR);              /*  5  */
      CALL CREATE$PROC(PTR);           /*  6  */
      CALL PREEMPT( BYT );             /*  7  */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*    CALL OUT$CHAR(BYT);              /*  8  */
/*    CALL OUT$LINE(PTR);              /*  9  */
/*    CALL OUT$NUM(BYT);               /* 10  */
/*    CALL OUT$DNUM(WORDS);            /* 11  */
/*    CALL IN$CHAR(PTR);               /* 12  */
/*    CALL IN$NUM(PTR);                /* 13  */
/*    CALL IN$DNUM(PTR);               /* 14  */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
   END;    /* CASE */
   RETURN;
END;    /* GATE$KEEPER */

/*0305**********************************************************/
/*  CREATE$EVC  PROCEDURE                        ROWE 6-22-84 */
/*-------------------------------------------------------------*/
/* CREATES EVENTCOUNT FOR INTER-PROCESS SYNCHRONIZATION.  */
/* EVENTCOUNT IS INITIALIZED TO 0 IN THE EVENTCOUNT TABLE.*/
/*************************************************************/
CREATE$EVC: PROCEDURE(NAME) REENTRANT PUBLIC;

DECLARE NAME BYTE;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* CALL OUT$LINE(@MSG21);
/* CALL OUT$NUM(NAME);
/* CALL OUT$CHAR(CR);
/* CALL OUT$CHAR(LF);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   /* ASSERT GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119); END;

   IF /* THE EVENTCOUNT DOES NOT ALREADY EXIST */
      LOCATE$EVC(NAME) = NOT$FOUND THEN DO;
      /* CREATE THE EVENTCOUNT ENTRY BY ADDING THE  */
      /* NEW EVENTCOUNT TO THE END OF THE EVC$TABLE */
      EVC$TBL(EVENTS).EVC$NAME = NAME;
      EVC$TBL(EVENTS).VALUE = 0;
      EVC$TBL(EVENTS).THREAD = 255;
```

```
      /* INCREMENT THE SIZE OF THE EVC$TABLE */
      EVENTS = EVENTS + 1;
   END; /* CREATE THE EVENTCOUNT */
   /* RELEASE THE GLOBAL LOCK */
   GLOBAL$LOCK = 0;
   RETURN;
END; /* CREATE$EVC PROCEDURE */


/*0356***************************************************************/
/*   READ PROCEDURE                                  ROWE 6-22-84  */
/*-------------------------------------------------------------------*/
/* THIS PROCEDURE ALLOWS USERS TO READ THE PRESENT VALUE  */
/* OF THE SPECIFIED EVENT$COUNT WITHOUT MAKING ANY        */
/* CHANGES.  A POINTER IS PASSED TO PROVIDE A BASE TO A   */
/* VARIABLE IN THE CALLING ROUTINE FOR PASSING THE RETURN */
/* VALUE BACK TO THE CALLING ROUTINE.                     */
/*******************************************************************/

READ:  PROCEDURE( EVC$NAME, RETS$PTR ) REENTRANT PUBLIC;

DECLARE
   EVC$NAME              BYTE,
   EVCTBL$INDEX          BYTE,
   RETS$PTR              POINTER,
   EVC$VALUE$RET         BASED RETS$PTR WORD;

   /*  SET THE GLOBAL LOCK  */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* CALL OUT$LINE(@MSG23);
/* CALL OUT$NUM(EVC$NAME);
/* CALL OUT$CHAR(CR);
/* CALL OUT$CHAR(LF);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   /*  OBTAIN INDEX  */
   EVCTBL$INDEX = LOCATE$EVC( EVC$NAME );

   /*  OBTAIN VALUE  */
   EVC$VALUE$RET = EVC$TBL( EVCTBL$INDEX ).VALUE;

   /*  UNLOCK GLOBAL LOCK  */
   GLOBAL$LOCK = 0 ;
   RETURN;
END;   /*  READ PROCEDURE  */


/*0412***************************************************************/
```

```
/*   AWAIT PROCEDURE                                              */
/*---------------------------------------------------------------*/
/* INTER PROCESS SYNCHRONIZATION PRIMITIVE.  SUSPENDS            */
/* EXECUTION OF RUNNING PROCESS UNTIL THE EVENTCOUNT HAS         */
/* REACHED THE SPECIFIED THRESHOLD VALUE, "AWAITED$VALUE."       */
/* USED BY THE OPERATING SYSTEM FOR THE MANAGEMENT OF            */
/* SYSTEM RESOURCES.                                             */
/*****************************************************************/

AWAIT: PROCEDURE(EVC$ID,AWAITED$VALUE) REENTRANT PUBLIC;

DECLARE
   AWAITED$VALUE     WORD,
  (EVC$ID, NEED$SCHED, RUNNING$VP,EVCTBL$INDEX) BYTE;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* CALL OUT$LINE(@MSG18);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   /* LOCK GLOBAL LOCK */
   DO WHILE LOCK$SET(@GLOBAL$LOCK, 119);  END;
   NEED$SCHED = TRUE;

   /*  DETERMINE THE RUNNING VIRTUAL PROCESSOR  */
   RUNNING$VP = RET$VP;

   /*  GET EVC INDEX  */
   EVCTBL$INDEX = LOCATE$EVC(EVC$ID);
   /* DETERMINE IF CURRENT VALUE IS LESS THAN THE
      AWAITED VALUE */
   IF EVC$TBL(EVCTBL$INDEX).VALUE < AWAITED$VALUE THEN DO;
      /* BLOCK PROCESS */
      VPM(RUNNING$VP).EVC$THREAD=EVC$TBL(EVCTBL$INDEX).THREAD;
      VPM(RUNNING$VP).EVC$AW$VALUE = AWAITED$VALUE;
      EVC$TBL( EVCTBL$INDEX ).THREAD = RUNNING$VP;
      DISABLE;
      PRDS.LAST$RUN = RUNNING$VP;
      VPM(RUNNING$VP).STATE = WAITING;
      END;     /* BLOCK PROCESS */

   ELSE /* DO NOT BLOCK PROCESS */
      NEED$SCHED = FALSE;

   /*  SCHEDULE THE VIRTUAL PROCESSOR  */
   IF NEED$SCHED = TRUE  THEN
      CALL VPSCHEDULER;        /* NO RETURN */

   /* UNLOCK GLOBAL LOCK */
   GLOBAL$LOCK = 0;
   RETURN;
```

```
END;    /* AWAIT PROCEDURE */


/*0482***************************************************/
/*    ADVANCE   PROCEDURE                  ROWE 6-22-84   */
/*---------------------------------------------------*/
/*  INTER PROCESS SYNCHRONIZATION PRIMITIVE.  INDICATES   */
/*  SPECIFIED EVENT HAS OCCURRED BY ADVANCING/INCREMENTING*/
/*  THE ASSOCIATED EVENTCOUNT.  EVENT IS BROADCAST TO ALL */
/*  VIRTUAL PROCESSORS AWAITING THAT EVENT.               */
/*---------------------------------------------------*/
/*  CALLS MADE TO:  OUT$LINE                             */
/*                  VPSCHEDULER (NO RETURN)              */
/***************************************************/

ADVANCE: PROCEDURE(EVC$ID) REENTRANT PUBLIC;

DECLARE
  (EVC$ID, NEED$SCHED, NEED$INTR, EVCTBL$INDEX) BYTE,
  (SAVE, RUNNING$VP, I, CPU)                    BYTE;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* CALL OUT$LINE(@MSG19);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   /* LOCK THE GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

   RUNNING$VP = RET$VP;
   EVCTBL$INDEX = LOCATE$EVC(EVC$ID);
   EVC$TBL(EVCTBL$INDEX).VALUE=EVC$TBL(EVCTBL$INDEX).VALUE + 1;
   NEED$SCHED = FALSE;
   NEED$INTR = FALSE;
   SAVE = 255;
   I = EVC$TBL( EVCTBL$INDEX ).THREAD;
   DO WHILE  I <> 255;
      IF VPM(I).EVC$AW$VALUE <= EVC$TBL(EVCTBL$INDEX).VALUE
         THEN DO;/* AWAKEN THE PROCESS */
         VPM(I).STATE = READY;
         VPM(I).EVC$AW$VALUE = 0;
         CPU = I / MAX$VPS$CPU ;
         IF SAVE = 255 THEN DO; /*THIS FIRST ONE IN LIST*/
            EVC$TBL(EVCTBL$INDEX).THREAD=VPM(I).EVC$THREAD;
            VPM( I ).EVC$THREAD = 255;
            I = EVC$TBL( EVCTBL$INDEX ).THREAD;
            END;  /* IF FIRST */
         ELSE DO;/* THEN THIS NOT FIRST IN LIST */
           VPM( SAVE ).EVC$THREAD = VPM( I ).EVC$THREAD;
            VPM( I ).EVC$THREAD = 255;
            I = VPM( SAVE ).EVC$THREAD;
```

```
         END;  /* IF NOT FIRST */
         IF  ( CPU <> PRDS.CPU$NUMBER ) THEN DO;
            HDW$INT$FLAG( CPU ) = TRUE;
            NEED$INTR = TRUE;
            END;
         ELSE NEED$SCHED = TRUE;
         END;     /* IF AWAKEN */
      ELSE DO;/* DO NOT AWAKEN THIS PROCESS */
         SAVE = I;
         I = VPM( I ).EVC$THREAD;
      END;/* IF NOT AWAKEN */
   END;/* DO WHILE */
   IF NEED$INTR = TRUE    THEN DO;  /* HARDWARE INTR */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*    CALL OUT$LINE( @MSG17 );
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
      DISABLE;
      OUTPUT(PORT$CA) =  80H;
      CALL TIME(1);
      OUTPUT(PORT$CA) = RESET;
      ENABLE;
   END;  /* NEED$INTR */
         IF NEED$SCHED = TRUE  THEN DO;
DISABLE;
      PRDS.LAST$RUN = RUNNING$VP;
      VPM(RUNNING$VP).STATE = READY;
      CALL VPSCHEDULER; /* NO RETURN */
   END;  /* IF NEED$SCHED */
   /* UNLOCK THE GLOBAL LOCK */
   GLOBAL$LOCK = 0;
   RETURN;
END;  /* ADVANCE  PROCEDURE */

/*0581***************************************************************/
/*  PREEMPT PROCEDURE                              ROWE 6-22-84   */
/*------- ---------------------------------------------------------*/
/* THIS PROCEDURE AWAKENS A HI PRIORITY PROCESS LEAVING           */
/* THE CURRENT RUNNING PROCESS IN THE  READY STATE AND            */
/* CALLS FOR A RESCHEDULING.  THE HIGH PRICRITY PROCESS           */
/* SHOULD BLOCK ITSELF WHEN FINISHED.                             */
/*   IF THE VP$ID IS 'FE' OR THE MONITOR PROCESS, IT WILL         */
/* MAKE IT READY WHEREVER IT IS IN THE VPM.  THE FOLLOW-          */
/* ING CODE DOES NOT TAKE ADVANTAGE OF THE FACT THAT              */
/* CURRENTLY IT IS THE THIRD ENTRY IN THE VPM FOR EACH            */
/* REAL PROCESSOR.                                                */
/*----------------------------------------------------------------*/
/* CALLS MADE TO:  OUTLINE, VPSCHEDULER                           */
/*****************************************************************/

PREEMPT:  PROCEDURE( VP$ID ) REENTRANT PUBLIC;
```

```
   DECLARE (VP$ID,SEARCH$ST,SEARCH$END,CPU,INDEX) BYTE;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* CALL OUT$LINE( @MSG16 );
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
   IF VP$ID <> 0FEH THEN DO; /* NORMAL PREEMPT */
     /* SEARCH VPM FOR INDEX FOR ID  */
      SEARCH$ST = 0;
      DO CPU = 0 TO (NR$RPS - 1);
         SEARCH$END = SEARCH$ST + NR$VPS( CPU ) - 1 ;
         DO INDEX = SEARCH$ST TO SEARCH$END;
            IF VPM( INDEX ).VP$ID = VP$ID  THEN GO TO FOUND;
         END; /* DO INDEX */
         SEARCH$ST = SEARCH$ST + MAX$VPS$CPU;
      END; /* DO CPU */
      /* CASE IF NOT FOUND IS NOT ACCOUNTED FOR CURRENTLY */
      FOUND:
         /* LOCK THE GLOBAL LOCK */
         DO WHILE LOCK$SET(@GLOBAL$LOCK,119);  END;
         /* SET PREEMPTED VP TO READY */
         VPM( INDEX ).STATE = READY;
         /* NEED HARDWARE INTR OR RE-SCHED  */
         IF ( CPU = PRDS.CPU$NUMBER ) THEN DO;
            INDEX = RET$VP;   /* DETERMINE RUNNING PROCESS */
            DISABLE;
            PRDS.LAST$RUN = INDEX;
            VPM( INDEX ).STATE = READY;  /* SET TO READY */
            CALL VPSCHEDULER;   /* NO RETURN */
            END;
         ELSE DO;/* CAUSE HARDWARE INTERRUPT */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*          CALL OUT$LINE(@MSG17);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
            HDW$INT$FLAG( CPU ) = TRUE;
            DISABLE;  OUTPUT( PORT$CA ) =  80H;
            CALL TIME(1);
            OUTPUT( PORT$CA ) = RESET;  ENABLE;
         END;
   END; /* NORMAL PREEMPT */
   ELSE DO; /* PREEMPT THE MONITOR */
      /* SEARCH VPM FOR ALL ID'S OF 0FEH */
      SEARCH$ST  = 0;
      DO WHILE LOCK$SET(@GLOBAL$LOCK,119); END;
      DO CPU = 0 TO (NR$RPS - 1);
         SEARCH$END = SEARCH$ST + NR$VPS( CPU ) - 1;
         /* SET ALL INT$FLAGS EXCEPT THIS CPU'S */
         IF PRDS.CPU$NUMBER <> CPU THEN
```

```
            HDW$INT$FLAG( CPU ) = TRUE;
         DO INDEX = SEARCH$ST TO SEARCH$END;
            IF VPM( INDEX ).VP$ID = VP$ID THEN
               VPM( INDEX ).STATE = READY;
         END; /* DO */
         SEARCH$ST = SEARCH$ST + MAX$VPS$CPU;
      END; /* ALL MONITOR PROCESS SET TO READY */
      /* INTERRUPT THE OTHER CPU'S AND
      RESCHEDULE THIS ONE                    */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*    CALL OUT$LINE(@MSG17);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
      DISABLE;
      OUTPUT( PORT$CA ) = 80H;
      CALL TIME(1);
      OUTPUT( PORT$CA ) = RESET;
      ENABLE;
      INDEX = RET$VP;
      DISABLE;
      PRDS.LAST$RUN = INDEX;
      VPM(INDEX).STATE = READY;
      CALL VPSCHEDULER;  /* NO RETURN */
   END; /* ELSE
   /* UNLOCK GLOBAL MEMORY */
   GLOBAL$LOCK = 0;
   RETURN;
 END;   /*  PREEMPT PROCEDURE  */


/*0711*************************************************************/
/*    CREATE$SEQ  PROCEDURE                  ROWE 6-22-84     */
/*-----------------------------------------------------------*/
/* CREATOR OF INTER PROCESS SEQUENCER PRIMITIVES FOR USER */
/* PROGRAMS.  CREATES A SPECIFIED SEQUENCER AND INITIAL-  */
/* IZES IT TO 0, BY ADDING THE SEQUENCER TO THE END OF THE*/
/* SEQUENCER TABLE.                                       */
/*-----------------------------------------------------------*/
/* CALLS MADE TO:  OUT$LINE          OUT$CHAR             */
/*                 OUT$HEX                                */
/*************************************************************/

CREATE$SEQ: PROCEDURE(NAME) REENTRANT PUBLIC;

DECLARE NAME BYTE;

   /* ASSERT GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119); END;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
```

```
/* CALL OUT$LINE(@MSG25);
/* CALL OUT$HEX(NAME);
/* CALL OUT$CHAR(CR);
/* CALL OUT$CHAR(LF);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   IF /* THE SEQUENCER DOES NOT ALREADY EXIST, IE */
      LOCATE$SEQ(NAME) = NOT$FOUND THEN DO;
      /* CREATE THE SEQUENCER ENTRY BY ADDING THE  */
      /* NEW SEQUENCER TO THE END OF THE SEQ$TABLE */
      SEQ$TABLE(SEQUENCERS).SEQ$NAME   = NAME;
      SEQ$TABLE(SEQUENCERS).SEQ$VALUE  =  0;
      /*  INCREMENT NUMBER OF SEQUENCERS  */
      SEQUENCERS = SEQUENCERS + 1;
   END; /* CREATE THE SEQUENCER */
   /* RELEASE THE GLOBAL LOCK */
   GLOBAL$LOCK = 0;
   RETURN;
END; /* CREATE$SEQ PROCEDURE */


/*0769***************************************************************/
/*  TICKET   PROCEDURE                          ROWE 6-22-84     */
/*-------------------------------------------------------------*/
/* INTER-VIRTUAL PROCESSOR SEQUENCER PRIMITIVE FOR  USER  */
/* PROGRAM.  SIMILAR TO "TAKE A NUMBER AND WAIT."  RETURNS*/
/* PRESENT VALUE OF SPECIFIED SEQUENCER AND INCREMENTS THE*/
/* SEQUENCER.  A POINTER IS PASSED TO PROVIDE A BASE TO A */
/* VARIABLE IN THE CALLING ROUTINE FOR PASSING THE RETURN */
/* VALUE BACK TO THE CALLING ROUTINE.                     */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$LINE                               */
/***************************************************************/

TICKET:  PROCEDURE( SEQ$NAME, RETS$PTR ) REENTRANT PUBLIC;

   DECLARE
      SEQ$NAME        BYTE,
      SEQTBL$INDEX    BYTE,
      RETS$PTR        POINTER,
      SEQ$VALUE$RET   BASED RETS$PTR WORD;

   /* ASSERT GLOBAL LOCK  */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* CALL OUT$LINE(@MSG24);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
```

```
   /*  OBTAIN SEQ$NAME INDEX  */
   SEQTBL$INDEX = LOCATE$SEQ( SEQ$NAME );
   /*  OBTAIN SEQUENCER VALUE  */
   SEQ$VALUE$RET = SEQ$TABLE( SEQTBL$INDEX ).SEQ$VALUE;
   /*  INCREMENT SEQUENCER  */
   SEQ$TABLE( SEQTBL$INDEX ).SEQ$VALUE  =
       SEQ$TABLE(SEQTBL$INDEX).SEQ$VALUE  +  1 ;

   /*  UNLOCK THE GLOBAL LOCK  */
   GLOBAL$LOCK  =  0 ;
   RETURN;
END;    /*  TICKET PROCEDURE  */


/*0828*************************************************************/
/*         CREATE$PROC    PROCEDURE        POWE  6-22-84    */
/*-----------------------------------------------------------*/
/*  THIS PROCEDURE CREATES A PROCESS FOR THE USER AS         */
/*  SPECIFIED BY THE INPUT PARAMETERS CONTAINED IN A         */
/*  STRUCTURE IN THE GATE MODULE.  THE PARAMETER PASSED      */
/*  IS A POINTER WHICH POINTS TO THIS STRUCTURE.             */
/*  INFO CONTAINED IN THIS STRUCTURE IS:  PROCESS ID,        */
/*  PROCESS PRIORITY, THE DESIRED PROC STACK LOCATION,       */
/*  AND THE PROCESS CODE STARTING LOCATION WHICH IS          */
/*  IS TWO ELEMENTS: THE IP REGISTER (OFFSET) AND THE        */
/*  CS REGISTER (CODE SEGMENT).                              */
/*-----------------------------------------------------------*/
/* CALLS MADE TO:   OUTLINE                                  */
/*************************************************************/

CREATE$PROC: PROCEDURE( PROC$PTR ) REENTRANT PUBLIC;

   DECLARE
      PROC$PTR        POINTER,
      PROC$TABLE BASED PROC$PTR STRUCTURE
        (PROC$ID              BYTE,
         PROC$PRI             BYTE,
         PROC$SP              WORD,
         PROC$SS              WORD,
         PROC$IP              WORD,
         PROC$CS              WORD,
         PROC$DS              WORD,
         PROC$ES              WORD);


   DECLARE
     (PS1, PS2)   WORD,
      TEMP        BYTE;

   DECLARE PROC$STACK$PTR POINTER AT(@PS1),
      PROC$STACK BASED PROC$STACK$PTR STRUCTURE
     (LENGTH(0FFH)      BYTE,
```

```
      RET$TYPE          WORD,
      BP                WORD,
      DI                WORD,
      SI                WORD,
      DS                WORD,
      DX                WORD,
      CX                WORD,
      AX                WORD,
      BX                WORD,
      ES                WORD,
      IP                WORD,
      CS                WORD,
      FL                WORD);

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* CALL OUT$LINE(@MSG26);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
   /*  TO SET UP PROC$STACK$PTR  */
   PS1 = PROC$TABLE.PROC$SP - 118H;
   PS2 = PROC$TABLE.PROC$SS;

   PROC$STACK.RET$TYPE = INT$RETURN;
   PROC$STACK.BP = PROC$TABLE.PROC$SP;
   PROC$STACK.DI = 0;
   PROC$STACK.SI = 0;
   PROC$STACK.DS = PROC$TABLE.PROC$DS;
   PROC$STACK.DX = 0;
   PROC$STACK.CX = 0;
   PROC$STACK.AX = 0;
   PROC$STACK.BX = 0;
   PROC$STACK.ES = PROC$TABLE.PROC$ES;
   PROC$STACK.IP = PROC$TABLE.PROC$IP;
   PROC$STACK.CS = PROC$TABLE.PROC$CS;
   PROC$STACK.FL = 200H;  /*SET IF FLAG (ENABLE INTR)*/

   /* SET GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

   IF PRDS.VPS$PER$CPU < MAX$VPS$CPU  THEN DO;
      TEMP = PRDS.VPS$PER$CPU + PRDS.VP$START;
      VPM( TEMP ).VP$ID = PROC$TABLE.PROC$ID;
      VPM( TEMP ).STATE = 01;/* READY */
      VPM( TEMP ).VP$PRIORITY = PROC$TABLE.PROC$PRI;
      VPM( TEMP ).EVC$THREAD = 255;
      VPM( TEMP ).EVC$AW$VALUE = 0;
      VPM( TEMP ).SP$REG = PROC$TABLE.PROC$SP - 1AH;
      VPM( TEMP ).SS$REG = PROC$TABLE.PROC$SS;

      PRDS.VPS$PER$CPU = PRDS.VPS$PER$CPU + 1;
      PRDS.VP$END = PRDS.VP$END + 1;
```

```
      NR$VPS( PRDS.CPU$NUMBER ) =
         NR$VPS(PRDS.CPU$NUMBER) + 1;
   END;    /* DO */

   /* RELEASE THE GLOBAL LOCK */
   GLOBAL$LOCK = 0;
   RETURN;
END; /*  CREATE$PROCESS  */


/*0958***************************************************************/
/*    IN$CHAR    PROCEDURE                          ROWE 6-22-84   */
/*---------------------------------------------------------------*/
/* GETS A CHAR FROM THE SERIAL PORT.  CHAR IS !!!NOT!!!          */
/* ECHOED.  THAT IS RESPONSIBILITY OF USER IN THIS CASE.         */
/* INPUT TO SERIAL PORT VIA SBC861 DOWN LOAD PROGRAM MAY         */
/* NOT BE ACCEPTED.                                              */
/* POINTER IS PROVIDED BY USER SO HE CAN BE RETURNED THE         */
/* CHARACTER .                                                   */
/*---------------------------------------------------------------*/
/* CALLS MADE TO:  RECV$CHAR                                     */
/*****************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* IN$CHAR:  PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;

/*    DECLARE
/*       RET$PTR POINTER,
/*       INCHR BASED RET$PTR BYTE;

/*    DISABLE;
/*    INCHR = RECV$CHAR;
/*    ENABLE;

/*    RETURN;
/* END; /* IN$CHAR */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

/*1002***************************************************************/
/*   IN$NUM      PROCEDURE                    ROWE  6-22-84       */
/*---------------------------------------------------------------*/
/*  GETS TWO ASCII CHAR FROM THE SERIAL PORT, EXAMINES           */
/*  THEM TO SEE IF THEY ARE IN THE SET 0. F HEX AND FORMS        */
/*  A BYTE VALUE.  EACH VALID HEX DIGIT IS ECHOED TO THE         */
/*  CRT.  IMPROPER CHAR ARE IGNORED.  NO ALLOWANCES ARE          */
/*  MADE FOR WRONG DIGITS.  GET IT RIGHT THE FIRST TIME.         */
/*  IF YOU ARE INDIRECTLY ACCESSING THE SERIAL PORT VIA          */
/*  THE SBC861 DOWN LOAD PROGRAM FROM THE MDS SYSTEM             */
/*  INPUT MAY NOT BE ACCEPTED.  A POINTER IS PASSED BY THE       */
/*  USER SO THAT HE RETURNED THE CHARACTER.                      */
```

```
/*-------------------------------------------------------*/
/*  CALLS MADE TO:  IN$HEX                               */
/*******************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* IN$NUM:  PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;
/*    DECLARE
/*       RET$PTR     POINTER,
/*       NUM BASED RET$PTR BYTE;

/*    DISABLE;
/*    NUM = IN$HEX;
/*    ENABLE;
/*    RETURN;
/* END; /* IN$NUM */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/


/*1034*****************************************************/
/*******************************************************/
/*   OUT$CHAR     PROCEDURE                  ROWE 6-22-84  */
/*-------------------------------------------------------*/
/* SENDS A BYTE TO THE SERIAL PORT                       */
/*-------------------------------------------------------*/
/* CALL MADE TO:   SEND$CHAR                             */
/*******************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* OUT$CHAR: PROCEDURE( CHAR ) REENTRANT PUBLIC;

/*    DECLARE CHAR BYTE;

/*    DISABLE;
/*    CALL SEND$CHAR( CHAR );
/*    ENABLE;
/*    RETURN;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/


/*1072*****************************************************/
/*   OUT$LINE PROCEDURE                  ROWE  6-22-84    */
/*-------------------------------------------------------*/
/* USING A POINTER TO A BUFFER IT WILL OUTPUT AN ENTIRE  */
/* LINE THRU THE SERIAL PORT UNTIL AN '%' IS ENCOUNTERED */
/* OR 80 CHARACTERS IS REACHED--WHICH EVER IS FIRST.  CR'S*/
/* AND LF'S CAN BE INCLUDED.                             */
/*-------------------------------------------------------*/
```

```
/* CALLS MADE TO:  SEND$CHAR                                          */
/*********************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* OUT$LINE: PROCEDURE( LINE$PTR ) REENTRANT PUBLIC;

/*    DECLARE
/*       LINE$PTR POINTER,
/*       LINE BASED LINE$PTR (80) BYTE,
/*       II BYTE;

/*    DISABLE;
/*    DO II = 0 TO 79;
/*       IF LINE( II ) = '%' THEN  GO TO DONE;
/*       CALL SEND$CHAR( LINE( II ) );
/*    END;
/*    DONE:   ENABLE;
/*    RETURN;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/


/*1104*********************************************************/
/*  OUT$NUM         PROCEDURE                ROWE   6-22-84   */
/*-------------------------------------------------------------*/
/*  OUTPUTS A BYTE VALUE NUMBER THRU THE SERIAL PORT           */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$HEX                                    */
/*********************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

/* OUT$NUM: PROCEDURE( NUM ) REENTRANT PUBLIC;

/*    DECLARE NUM BYTE;

/*    DISABLE;
/*    CALL OUT$HEX( NUM );
/*    ENABLE;
/*    RETURN;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/


/*1142*********************************************************/
/*  IN$DNUM        PROCEDURE                ROWE  6-22-84    */
/*-------------------------------------------------------------*/
/* GETS FOUR ASCII FROM SERIAL PORT TO FORM WORD VALUE.        */
```

```
/* CRITERIA ARE THE SAME AS IN PROCEDURE IN$NUM                        */
/*--------------------------------------------------------------------*/
/* CALLS MADE TO:  IN$HEX                                             */
/**********************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* IN$DNUM:  PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;

/*    DECLARE
/*       RET$PTR     POINTER,
/*       DNUM BASED RET$PTR WORD,
/*      (H, L)   WORD;

/*    DISABLE;
/*    H = IN$HEX;
/*    H = SHL( H, 8 );
/*    L  = IN$HEX;
/*    DNUM = (H OR L);
/*    ENABLE;
/*    RETURN;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/


/*1172**************************************************************/
/*    OUT$DNUM     PROCEDURE                ROWE 6-22-84          */
/*--------------------------------------------------------------*/
/*  OUTPUTS A WORD VALUE NUMBER VIA THE SERIAL PORT              */
/*--------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$HEX                                     */
/****************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*  OUT$DNUM: PROCEDURE( DNUM ) REENTRANT PUBLIC;

/*    DECLARE
/*       DNUM    WORD,
/*       SEND    BYTE;

/*    DISABLE;
/*    SEND = HIGH( DNUM );
/*    CALL OUT$HEX( SEND );
/*    SEND = LOW( DNUM );
/*    CALL OUT$HEX( SEND );
/*    ENABLE;
/*    RETURN;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
```

```
/*1215*************************************************************/
/*   RECV$CHAR   PROCEDURE                    ROWE 6-22-84       */
/*-----------------------------------------------------------------*/
/* BOTTOM LEVEL PROCEDURE THAT OBTAINS A CHAR FROM THE           */
/* SERIAL PORT.  PARITY BIT IS REMOVED.  CHAR IS !!NOT!!         */
/* ECHOED.                                                       */
/*-----------------------------------------------------------------*/
/* CALLS MADE TO:    NONE                                        */
/*************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* RECV$CHAR:  PROCEDURE BYTE REENTRANT PUBLIC;

/*    DECLARE
/*       CHR     BYTE;

/*    /*CHECK PORT STATUS BIT 2 FOR RECEIVE-READY SIGNAL */
/*    DO WHILE (INPUT(0DAH) AND 02H) = 0;  END;
/*    CHR = (INPUT(0D8H) AND 07FH);
/*    RETURN CHR;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/


/*1241*************************************************************/
/*  SEND$CHAR     PROCEDURE                 ROWE     6-22-84   */
/*-----------------------------------------------------------------*/
/*  OUTPUTS A BYTE THRU THE SERIAL PORT.  THIS IS NOT A          */
/*  SERVICE AVAILABLE THRU THE GATEKEEPER BUT IT IS CALLED*/
/*  BY MANY OF THOSE PROCEDURES.  IT WILL STOP SENDING           */
/*  (AND EVERYTHING ELSE) IF IT SEES A ^S AT INPUT.  ^Q          */
/*  WILL RELEASE THE PROCEDURE TO CONTINUE.                      */
/*  THE USER BEWARE!!!!! THIS IS ONLY A DIAGNOSTIC TOOL          */
/*  TO FREEZE THE CRT FOR STUDY.  RELEASING IT DOESN'T    */
/*  ASSURE NORMAL RESUMPTION OF EXECUTION.  (YOU MAY FORCE*/
/*  ALL BOARDS TO IDLE FOR EXAMPLE.)                             */
/*-----------------------------------------------------------------*/
/*  CALLS MADE TO:                                               */
/*************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* SEND$CHAR: PROCEDURE(CHAR) REENTRANT PUBLIC;

/*    DECLARE (CHAR,INCHR) BYTE;

/*    /* CHECK PORT STATUS */
/*    INCHR = (INPUT(0D8H) AND 07FH);
```

```
/*    IF INCHR = 13H THEN
/*       DO WHILE (INCHR <> 11H);
/*          IF ((INPUT(0DAH) AND 02H) <> 0) THEN
/*             INCHR = (INPUT(0D8H) AND 07FH);
/*       END;
/*    DO WHILE (INPUT(0DAH) AND 01H) = 0;   END;
/*    OUTPUT(0D8H) = CHAR;
/*    RETURN;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/


/*1293*************************************************************/
/*  IN$HEX    PROCEDURE                          ROWE 6-22-84   */
/*-----------------------------------------------------------*/
/*  GETS 2 HEX CHAR FROM THE SERIAL PORT AND IGNORES ANY-  */
/*  THING ELSE.  EACH VALID HEX DIGIT IS ECHOED TO THE     */
/*  SERIAL PORT.  A BYTE VALUE IS FORMED FROM THE TWO HEX  */
/*  CHAR.                                                  */
/*-----------------------------------------------------------*/
/*  CALLS MADE TO:  RECV$CHAR                              */
/*************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* IN$HEX:  PROCEDURE  BYTE REENTRANT PUBLIC;

/*    DECLARE
/*       ASCII(*) BYTE DATA ('0123456789ABCDEF'),
/*    ASCIIH(*) BYTE DATA('0123456789',61H,62H,63H,64H,65H,
/*          66H),

/*      (INCHR, HEXNUM, H, L)  BYTE,
/*       FOUND                 BYTE,
/*       STOP                  BYTE;

/*    /* GET HIGH PART OF BYTE */
/*    FOUND = 0;
/*    DO WHILE NOT FOUND;
/* /* IF INVALID CHAR IS INPUT, COME BACK HERE */
/*       INCHR = RECV$CHAR;
/*       H = 0;
/*       STOP = 0;
/* /* COMPARE CHAR TO HEX CHAR SET */
/*       DO WHILE NOT STOP;
/*       IF (INCHR=ASCII(H)) OR (INCHR = ASCIIH(H)) THEN DO;
/*             STOP = 0FFH;
/*             FOUND = 0FFH;
/*             CALL SEND$CHAR( INCHR ); /* TO ECHO IT */
/*             END;
/*          ELSE DO;
```

```
/*              H = H + 1;
/*              IF H = 10H THEN STOP = 0FFH;
/*           END; /* ELSE */
/*        END; /* DO WHILE */
/*        H = SHL( H, 4 );
/*     END; /* DO WHILE */
/*     FOUND = 0;
/*     /* GET LOW PART OF BYTE */
/*     DO WHILE NOT FOUND;
/* /* AGAIN DO UNTIL VALID HEX CHAR IS INPUT */
/*        INCHR = RECV$CHAR;
/*        L = 0H;
/*        STOP = 0;
/*        DO WHILE NOT STOP;
/*        IF (INCHR=ASCII(L)) OR (INCHR=ASCIIH(L)) THEN DO:
/*              STOP = 0FFH;
/*              FOUND = 0FFH;
/*              CALL SEND$CHAR(INCHR);
/*              END;
/*           ELSE DO;
/*              L = L + 1;
/*              IF L = 10H THEN STOP = 0FFH;
/*           END; /* ELSE */
/*        END; /* DO WHILE */
/*     END; /* DO WHILE */
/*     RETURN (H OR L);
/* END; /* IN$HEX */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

/*1393*************************************************************/
/*     OUT$HEX    PROCEDURE                        ROWE 6-22-84 */
/*-------------------------------------------------------------*/
/*  TRANSLATES BYTE VALUES TO ASCII CHARACTERS AND OUTPUTS*/
/*  THEM THRU THE SERIAL PORT                                  */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO:  SEND$CHAR                                  */
/*************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/* OUT$HEX: PROCEDURE(B) REENTRANT PUBLIC;

/*     DECLARE B BYTE;
/*     DECLARE ASCII(*) BYTE DATA ('0123456789ABCDEF');

/*     CALL SEND$CHAR(ASCII(SHR(B,4) AND 0FH));
/*     CALL SEND$CHAR(ASCII(B AND 0FH));
/*     RETURN;
/* END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
```

```
/*****************************************************************/

END;    /*  L2$MODULE  */

/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
```

APPENDIX H

LEVEL I -- MCORTEX SOURCE CODE

All the LEVEL I source code written in PL/M is contained in the file LEVEL1.SRC. It is compiled with the LARGE attribute. Two other LEVEL I functions, SCHEDULER and INTERRUPT HANDLER, were written in ASM86 and are listed in their own modules. LEVEL I is one of the relocateable code modules in file: KORE.LNK. It is part of the executable code module in file: KORE. KORE is the development system version of the file KORE.OPS loaded by MCORTEX.CMD under the CP/M-86 operating system. This module contains utility procedures used only by the operating system. Two memory maps of KORE (.OPS and .TRC) are located at the end of this Appendix. The maps come from file: KORE.MP2 after compiling, linking and locating the applicable files. KORE(OPS) is produced with the code unaltered. KOPE(TRC) is obtained by removing and adding appropriate comment marks from the indicated code before processing.

```
/*0027*************************************************************/
/*************************************************************/
/*************************************************************/
/*  FILE:           LEVEL1.SRC
    VERSION:        ROWE 6-22-84
    PROCEDURES
        DEFINED:    RET$VP          RDYTHISVP
                    GETWORK         LOCATE$EVC
                    LOCATE$SEQ      IDLE$PROC
                    SAVE$CONTEXT    GET$SP
                    MONITOR$PROC

    REMARKS:
       WARNING:  SEVERAL OF THE LITERAL DECLARATIONS BELOW
       HAVE A SIMILAR MEANING IN OTHER MODULES.  THAT MEAN-
       ING IS COMMUNICATED ACROSS MODULES BOUNDARIES.  BE
       CAREFUL WHEN CHANGING THEM.
                                                            */
/*************************************************************/

L1$MODULE:  DO;

/*0029*************************************************************/
/*************************************************************/
/*  LOCAL DECLARATIONS                                       */

DECLARE
   MAX$CPU                    LITERALLY      '10',
   MAX$VPS$CPU                LITERALLY      '10',
   MAX$CPU$$$MAX$VPS$CPU      LITERALLY     '100',
   FALSE                      LITERALLY       '0',
   READY                      LITERALLY       '1',
   RUNNING                    LITERALLY       '3',
   WAITING                    LITERALLY       '7',
   TRUE                       LITERALLY     '119',
   NOT$FOUND                  LITERALLY     '255',
   PORT$C0                    LITERALLY  '00C0H',
   PORT$C2                    LITERALLY  '00C2H',
   PORT$CE                    LITERALLY  '00CEH',
   PORT$CA                    LITERALLY  '00CAH',
   RESET                      LITERALLY       '0',
   INT$RETURN                 LITERALLY    '77H',

/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
   IDLE$STACK$SEG             LITERALLY  '0C80H',    /********/
   IDLE$STACK$ABS             LITERALLY '0C800H',    /********/
   INIT$STACK$SEG             LITERALLY  '0C88H',    /********/
   INIT$STACK$ABS             LITERALLY '0C880H';    /********/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
```

```
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* IDLE$STACK$SEG          LITERALLY  '0C50H',    /*********/
/* IDLE$STACK$ABS          LITERALLY '0C500H',    /*********/
/* INIT$STACK$SEG          LITERALLY  '0C58H',    /*********/
/* INIT$STACK$ABS          LITERALLY '0C580H',    /*********/
/* MONITOR$STACK$SEG       LITERALLY  '0C60H',    /*********/
/* MONITOR$STACK$ABS       LITERALLY '0C600H';    /*********/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

/*0084**************************************************************/
/*  PROCESSOR DATA SEGMENT TABLE                                    */
/*    INFORMATION RELEVANT TO THE PARTICULAR PHYSICAL               */
/*    PROCESSOR ON WHICH IT IS RESIDENT.                            */
/*                                                                  */
/*    CPU$NUMBER:     UNIQUE SEQUENTIAL NUMPER ASSIGNED TO          */
/*                    THIS REAL PROCESSOR.                          */
/*    VP$START:       VPM INDEX OF THE FIRST VIRTUAL                */
/*                    PROCESS ASSIGNED TO THIS REAL CPU.            */
/*    VP$END:         INDEX IN VPM OF LAST VIRTUAL...               */
/*    VPS$PER$CPU:    THE NUMBER OF VP ASSIGNED TO THIS             */
/*                    REAL CPU.  MAX IS 10.                         */
/*    LAST$RUN:       VPM INDEX OF THE PROCESS MOST                 */
/*                    RECENTLY SWITCHED FROM RUNNING TO             */
/*                    EITHER READY OR WAITING.                      */
/*    COUNTER:        AN ARBITRARY MEASURE OF PERFORMANCE.          */
/*                    COUNT MADE WHILE IN IDLE STATE.               */

DECLARE PRDS STRUCTURE
  (CPU$NUMBER       BYTE,
   VP$START         BYTE,
   VP$END           BYTE,
   VPS$PER$CPU      BYTE,
   LAST$RUN         BYTE,
   COUNTER          WORD) PUBLIC INITIAL(0,0,0,0,0,0);

/*0110**************************************************************/
/* GLOBAL DATA BASE DECLARATIONS                                    */
/*    DECLARED PUBLIC IN FILE   'GLOBAL.SRC'                        */
/*                      IN MODULE 'GLOBAL$MODULE'                   */

DECLARE VPM( MAX$CPU$$$$MAX$VPS$CPU ) STRUCTURE
  (VP$ID            BYTE,
   STATE            BYTE,
   VP$PRIORITY      BYTE,
   EVC$THREAD       BYTE,
   EVC$AW$VALUE     WORD,
   SP$REG           WORD,
   SS$REG           WORD) EXTERNAL;

DECLARE
   CPU$INIT         BYTE EXTERNAL,
```

```
   HDW$INT$FLAG( MAX$CPU ) BYTE EXTERNAL,
   NR$VPS( MAX$CPU ) BYTE EXTERNAL,
   NR$RPS          BYTE EXTERNAL,
   GLOBAL$LOCK     BYTE EXTERNAL;

DECLARE
   EVENTS BYTE EXTERNAL,
   EVC$TBL(100) STRUCTURE
      (EVC$NAME       BYTE,
       VALUE          WORD,
       THREAD         BYTE) EXTERNAL;

DECLARE
   SEQUENCERS BYTE EXTERNAL,
   SEQ$TABLE(100) STRUCTURE
       (SEQ$NAME      BYTE,
        SEQ$VALUE     WORD) EXTERNAL;

/*0159**************************************************************/
/* DECLARATION OF EXTERNAL PROCEDURE REFERENCES                     */
/*    THE FILE AND MODULE WHERE THEY ARE DEFINED ARE                */
/*    LISTED.                                                       */


INITIAL$PROC: PROCEDURE EXTERNAL;  END;
   /* IN FILE:    INITKK.SPC */
   /* IN MODULE:  INIT$MOD   */

AWAIT: PROCEDURE (EVC$ID,AWAITED$VALUE) EXTERNAL;
   DECLARE EVC$ID BYTE, AWAITED$VALUE WORD;
END;

VPSCHEDULER: PROCEDURE EXTERNAL;  END;
   /* IN FILE:    SCHED.ASM */

DECLARE INTVEC LABEL EXTERNAL;
   /* IN FILE:    SCHED.ASM */

DECLARE INTR$VECTOR POINTER AT(0110H) INITIAL(@INTVEC);
   /* IN FILE:    SCHED.ASM */

/*0182**************************************************************/
/*  THESE DIAGNOSTIC MESSAGES MAY EVENTUALLY BE REMOVED.  */
/*  THE UTILITY PROCEDURES, HOWEVER, ARE ALSO USED BY THE */
/*  MONITOR PROCESS.  THEY SHOULD NOT BE REMOVED.         */

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* DECLARE
/* MSG1(*)  BYTE INITIAL ('ENTERING RET$VP  ',13,10,'%'),
/* MSG1A(*) BYTE INITIAL ('   RUNNING$VP$INDEX =  %'),
/* MSG4(*)  BYTE INITIAL ('ENTERING RDYTHISVP',13,10,'%'),
```

```
/* MSG4A(*) BYTE INITIAL ('   SET VP TO READY:    VP =  %'),
/* MSG7(*)  BYTE INITIAL ('ENTERING GETWORK',13,10,'%'),
/* MSG7A(*) BYTE INITIAL ('   SET VP TO RUNNING:  VP =  %'),
/* MSG7B(*) BYTE INITIAL ('   SELECTED$DPR = %'),
/* MSG10(*) BYTE INITIAL ('ENTERING IDLE$VP ',13,10,'%'),
/* MSG11(*) BYTE INITIAL ('UPDATE IDLE COUNT ',13,10,'%'),
/* MSG12(*) BYTE INITIAL ('ENTERING KERNEL$INIT',10,13,'%'),
/* MSG20(*) BYTE INITIAL ('ENTERING LOCATE$EVC ',10,13,'%'),
/* MSG22(*) BYTE INITIAL ('ENTERING LOCATE$SEQ ',10,13,'%'),
/* MSG23(*) BYTE INITIAL ('   FOUND',10,13,'%'),
/* MSG24(*) BYTE INITIAL ('   NOT FOUND',10,13,'%');
/*
/*DECLARE
/*    CR LITERALLY '0DH',
/*    LF LITERALLY '0AH';
/*
/*OUT$CHAR: PROCEDURE( CHAR ) EXTERNAL;
/*    DECLARE CHAR BYTE;
/*END;
/*
/*OUT$LINE: PROCEDURE( LINE$PTR ) EXTERNAL;
/*    DECLARE LINE$PTR POINTER;
/*END;
/*
/*OUT$NUM: PROCEDURE( NUM ) EXTERNAL:
/*    DECLARE NUM BYTE;
/*END;
/*
/*OUT$DNUM: PROCEDURE( DNUM ) EXTERNAL;
/*    DECLARE DNUM WORD;
/*END;
/*
/*OUT$HEX: PROCEDURE(B) EXTERNAL;
/*    DECLARE B BYTE;
/*END;
/*
/*IN$CHAR: PROCEDURE ( RET$PTR )  EXTERNAL;
/*   DECLARE RET$PTR POINTER;
/*END;
/*
/*IN$DNUM:  PROCEDURE (RET$PTR) EXTERNAL;
/*   DECLARE RET$PTR POINTER;
/*END;
/*
/*IN$NUM: PROCEDURE (RET$PTR) EXTERNAL;
/*   DECLARE RET$PTR POINTER;
/*END;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

/*0273***************************************************/
/* STACK DATA & INITIALIAZTION FOR SYSTEM PROCESSES      */
```

```
DECLARE IDLE$STACK  STRUCTURE
     (LENGTH(030H)    WORD,
     RET$TYPE         WORD,
     BP               WORD,
     DI               WORD,
     SI               WORD,
     DS               WORD,
     DX               WORD,
     CX               WORD,
     AX               WORD,
     BX               WORD,
     ES               WORD,
     START           POINTER,  /* IP,CS */
     FL               WORD) AT(IDLE$STACK$ABS)
          INITIAL(
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 INT$RETURN,7AH,0,0,0,0,0,0,0,0,@IDLE$PROC,200H );

 DECLARE INIT$STACK  STRUCTURE
     (LENGTH(030H)    WORD,
     RET$TYPE         WORD,
     BP               WORD,
     DI               WORD,
     SI               WORD,
     DS               WORD,
     DX               WORD,
     CX               WORD,
     AX               WORD,
     BX               WORD,
     ES               WORD,
     START           POINTER,   /* IP,CS */
     FL               WORD) AT(INIT$STACK$ABS)
        INITIAL(
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
 INT$RETURN,7AH,0,0,0,0,0,0,0,0,@INITIAL$PROC,200H );
                     /* 200H SETS THE IF FLAG */

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/*  DECLARE MONITOR$STACK STRUCTURE
/*(LENGTH(030H)     WORD,
/*       RET$TYPE          WORD,
/*       BP                WORD,
/*       DI                WORD,
/*       SI                WORD,
/*       DS                WORD,
/*       DX                WORD,
/*       CX                WORD,
```

```
/*       AX                    WORD,
/*       BX                    WORD,
/*       ES                    WORD,
/*       START                 POINTER,    /* IP,CS */
/*       FL                    WORD) AT(MONITOR$STACK$ABS)
/*          INITIAL(
/* 0,0,0,2,0,0,2,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,
/* 0,0,0,0,2,2,0,0,2,0,0,2,0,0,0,0,0,0,0,0,2,0,
/*  INT$RETURN,7AF,0,0,0,0,0,0,0,0,@MONITOR$PROC,200H );
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

/*0354***************************************************************/
/*********************************************************************/
/*  RET$VP   PROCEDURE                              ROWE 8-22-84   */
/*-------------------------------------------------------------------*/
/* USED BY THE SCHEDULER TO FIND OUT WHAT IS THE CURRENT  */
/* RUNNING PROCESS.  IT'S INDEX IN VPM IS RETURNED.       */
/*-------------------------------------------------------------------*/
/* CALLS MADE TO:  OUT$HEX      OUT$CHAR                  */
/*********************************************************************/

RET$VP: PROCEDURE BYTE REENTRANT PUBLIC;

   DECLARE RUNNING$VP$INDEX BYTE;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG1);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

   /*  SEARCH THE VP MAP FOR RUNNING PROCESS INDEX */
   DO   RUNNING$VP$INDEX = PRDS.VP$START TO PRDS.VP$END;
      IF VPM( RUNNING$VP$INDEX ).STATE = RUNNING
      THEN GO TO FOUND;
   END; /* DO */
   RUNNING$VP$INDEX = PRDS.LAST$RUN;

FOUND:
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG1A);
/* CALL OUT$HEX(RUNNING$VP$INDEX);
/* CALL OUT$CHAR(CR);
/* CALL OUT$CHAR(LF);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   RETURN RUNNING$VP$INDEX;
END; /* RET$VP PROCEDURE */
```

```
/*0410***************************************************************/
/*  RDYTHISVP     PROCEDURE                    ROWE 6-22-84    */
/*-------------------------------------------------------------*/
/*  CHANGES A VIRTUAL PROCESSOR STATE TO READY                 */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$HEX    OUT$CHAR                        */
/***************************************************************/

RDYTHISVP: PROCEDURE REENTRANT PUPLIC;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG4);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

   PRDS.LAST$RUN = RET$VP;  /* SAVE THIS PROCESSOR INDEX */

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG4A);
/* CALL OUT$HEX(PRDS.LAST$RUN);
/* CALL OUT$CHAR(CR);
/* CALL OUT$CHAR(LF);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   VPM(PRDS.LAST$RUN).STATE = READY;
   RETURN;
END; /* RDYTHISVP PROCEDURE */


/*0441***************************************************************/
/*  SAVECONTEXT PROCEDURE                    ROWE  2 APR 84 */
/*-------------------------------------------------------------*/
/*  SAVES CURRENT STACK POINTER AND SEGMENT IN VPM             */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO: RET$VP                                      */
/***************************************************************/

SAVECONTEXT: PROCEDURE (STACK$PTR, STACK$SEG) REENTRANT
                                                    PUBLIC;

  DECLARE (STACK$PTR, STACK$SEG) WORD;

  IF PRDS.LAST$RUN <> 255 THEN DO; /* IF ENTRY IS NOT    */
                                   /* FROM KORE START    */
    VPM(PRDS.LAST$RUN).SP$REG = STACK$PTR; /* SAVE STACK */
    VPM(PRDS.LAST$RUN).SS$REG = STACK$SEG; /* STATE      */
  END;

END;
```

```
/*0480*************************************************************/
/*  GET$SP     PROCEDURE                          ROWE  2 APR 84 */
/*-------------------------------------------------------------*/
/*  RETURNS STACK POINTER OF CURRENT RUNNING PROCESS AS        */
/*  SAVED IN THE VIRTUAL PROCESSOR MAP                         */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO: RET$VP                                      */
/*************************************************************/

GET$SP: PROCEDURE WORD REENTRANT PUBLIC;

  DECLARE N BYTE;

  N = RET$VP;  /* GET CURRENT RUNNING VIRTUAL PROCESSOR    */

  RETURN VPM(N).SP$REG; /* RETURN NEW VP STACK POINTER     */

END;


/*0498*************************************************************/
/*  GETWORK    PROCEDURE                          ROWE  6-22-84  */
/*-------------------------------------------------------------*/
/*  DETERMINES THE NEXT ELIGIBLE VIRTUAL PROCESSOR TO RUN      */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$CHAR   OUT$LINE   OUT$DNUM             */
/*************************************************************/

GETWORK: PROCEDURE WORD REENTRANT PUBLIC;

   DECLARE (PRI,N,I)    BYTE;
   DECLARE SELECTED$DBR WORD;
   DECLARE DISPLAY      BYTE;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG7);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

   PRI = 255;
   DO /* SEARCH VPM FOR ELIGIBLE VIRTUAL PROCESSOR TO RUN */
      I = PRDS.VP$START TO PRDS.VP$END;
      IF /* THIS VP'S PRIORITY IS HIGHER THAN PRI */
         ((VPM(I).VP$PRIORITY <= PRI) AND
         (VPM(I).STATE = READY))  THEN  DO;
            /* SELECT THIS VIRTUAL PROCESSOR */
            PRI = VPM(I).VP$PRIORITY;
            N = I;
      END; /* IF */
```

```
   END; /* DO LOOP SEARCH OF VPM */

   /*  SET SELECTED VIRTUAL PROCESSOR  */
   VPM(N).STATE = RUNNING;
   SELECTED$DBR = VPM(N).SS$REG;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG7A);
/* CALL OUT$HEX(N);
/* CALL OUT$CHAR(CR);
/* CALL OUT$CHAR(LF);
/* CALL OUT$LINE(@MSG7B);
/* CALL OUT$DNUM(SELECTED$DBR);
/* CALL OUT$CHAR(CR);
/* CALL OUT$CHAR(LF);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

   RETURN SELECTED$DBR;

END; /* GETWORK PROCEDURE */


/*0568***************************************************************/
/*  LOCATE$EVC PROCEDURE                          ROWE 6-22-84    */
/*-------------------------------------------------------------------*/
/* FUNCTION CALL.  RETURNS THE INDEX IN EVENTCOUNT TABLE  */
/* OF THE EVENT NAME PASSED TO IT.                        */
/*-------------------------------------------------------------------*/
/* CALLS MADE TO:  OUT$CHAR   OUT$LINE                    */
/***************************************************************/

LOCATE$EVC: PROCEDURE(EVENT$NAME) BYTE REENTRANT PUBLIC;

   DECLARE EVENT$NAME BYTE;
   DECLARE (MATCH,EVCTBL$INDEX) BYTE;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG20);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

   MATCH = FALSE;
   EVCTBL$INDEX = 0;
   /* SEARCH DOWN THE EVENTCOUNT TABLE TO LOCATE THE  */
   /* DESIRED EVENTCOUNT BY MATCHING THE NAMES */
   DO WHILE (MATCH = FALSE)  AND  (EVCTBL$INDEX < EVENTS);
/* DO WHILE HAVE NOT FOUND THE EVENTCOUNT AND HAVE NOT */
/* REACHED END OF THE TABLE */
IF EVENT$NAME = EVC$TBL(EVCTBL$INDEX).EVC$NAME THEN
```

```
   MATCH = TRUE;
      ELSE
         EVCTBL$INDEX = EVCTBL$INDEX+1;
   END; /* WHILE */
   /* IF HAVE FOUND THE EVENTCOUNT */
   IF (MATCH = TRUE) THEN DO;
     /* RETURN ITS INDEX IN THE EVC$TBL */
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/*   CALL OUT$LINE(@MSG23);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
     RETURN EVCTBL$INDEX;
     END;
   ELSE DO;
   /* RETURN NOT FOUND CODE */
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/*   CALL OUT$LINE(@MSG24);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
     RETURN NOT$FOUND;
   END; /* ELSE */
END; /* LOCATE$EVC PROCEDURE */


/*0637*************************************************************/
/*  LOCATE$SEQ PROCEDURE                          POWE 6-22-84 */
/*-------------------------------------------------------------*/
/* FUNCTION CALL TO RETURN THE INDEX OF THE SEQUENCER          */
/* SPECIFIED IN THE SEQ-TABLE.                                 */
/*-------------------------------------------------------------*/
/* CALLS MADE TO:  OUT$LINE                                    */
/*************************************************************/

LOCATE$SEQ:  PROCEDURE(SEQ$NAME) BYTE REENTRANT PUBLIC;

   DECLARE SEQ$NAME BYTE;
   DECLARE ( MATCH, SEQTBL$INDEX ) BYTE;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG22);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

  MATCH = FALSE;
  SEQTBL$INDEX = 0;
  DO WHILE (MATCH = FALSE) AND (SEQTBL$INDEX < SEQUENCERS);
    IF SEQ$NAME = SEQ$TABLE(SEQTBL$INDEX).SEQ$NAME THEN
      MATCH = TRUE;
    ELSE
      SEQTBL$INDEX = SEQTBL$INDEX + 1;
```

```
  END; /* WHILE */
  IF (MATCH = TRUE) THEN DO;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/*  CALL OUT$LINE(@MSG23);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
    RETURN SFOTBL$INDEX;
    END; /* IF */
  ELSE DO;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/*  CALL OUT$LINE(@MSG24);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
    RETURN NOT$FOUND;
    END; /* ELSE */
END; /* LOCATE$SEQ PROCEDURE */


/*0698***************************************************************/
/*****************************************************************/
/*  SYSTEM PROCESSES                                             */
/*                                                               */


/*****************************************************************/
/*  IDLE PROCESS                                  ROWE  6-22-84  */
/*---------------------------------------------------------------*/
/*  THIS PROCESS IS SCHEDULED IF ALL OTHER PROCESSES IN          */
/*  THE VPM ARE BLOCKED.  THE STARTING ADDRESS IS PROVIDED       */
/*  TO THE IDLE$STACK AND PLACED IN PRDS.IDLE$DBR.  A            */
/*  COUNTER IS INCREMENTED ABOUT EVERY SECOND.  THE COUNT        */
/*  IS MAINTAINED IN THE PRDS TABLE AND IS A ROUGH MEASURE       */
/*  OF SYSTEM PERFORMANCE BY GIVING AN INDICATION OF THE         */
/*  AMOUNT OF TIME SPENT IN THE IDLE PROCESS.                    */
/*---------------------------------------------------------------*/
/*  CALLS MADE TO:  PLM86 PROCEDURE 'TIME'                       */
/*                  OUT$LINE                                     */
/*****************************************************************/
IDLE$PROC:  PROCEDURE REENTRANT PUBLIC;

   DECLARE I BYTE;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG10);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

   /*  DELAYS ONE (1) SECOND  */
LOOP: DO I = 1 TO 40;
```

```
   CALL TIME( 250 );
END;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/*    CALL OUT$LINE(@MSG11);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

      PRDS.COUNTER = PRDS.COUNTER + 1;
      GO TO LOOP;
END;   /* IDLE$PROC  */


/*0759**************************************************************/
/*  MONITOR PROCESS                              ROWE 6-22-84   */
/*--------------------------------------------------------------*/
/*  THE MONITOR PROCESS IS INITIALIZED BY THE OS LIKE           */
/*  INIT AND IDLE.  IT HAS THE RESERVED ID OF 0FFH AND A        */
/*  PRIORITY OF 0H.  IT IS ALWAYS BLOCKED OR WAITING UNTIL      */
/*  IT IS PREEMPTED BY THE USER.                                */
/*--------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$LINE       OUT$CHAR                     */
/*                  OUT$DNUM       IN$DNUM                      */
/*                  IN$NUM                                      */
/**************************************************************/

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* MONITOR$PROC:  PROCEDURE REENTRANT PUBLIC;
/*
/* DECLARE
/*    PTR                POINTER,
/*    PTR2               POINTER,
/*    PTR3 BASED PTR2    POINTER,
/*    ADDR STRUCTURE (OFFSET WORD, BASE WORD),
/*    CONTENTS BASED PTR BYTE;

/* DECLARE
/*   (LINECOMPLETE, LOOP2)     BYTE,
/*   (QUANTITY, COUNT)   BYTE,
/*   (INCHR, INDEX, VALID$CMD) BYTE;

/* LOOP:  VALID$CMD = 0;

/*    CALL OUT$CHAR(CR);
/*    CALL OUT$CHAR(LF);
/*    CALL OUT$CHAR('.');
/*    DO WHILE NOT VALID$CMD;
/*       CALL INSCHAR(@INCHR);
/*  IF (INCHR = 'D') OR (INCHR = 'S') OR (INCHR = 'E') THEN
/*     VALID$CMD = 0FFH;
```

```
/*  IF (INCHR=64H) OR (INCHR=65H) OR (INCHR=73H) THEN
/*     VALID$CMD = 0FFH;
/*  IF VALID$CMD = 0FFH THEN CALL OUT$CHAR(INCHR);
/*    END; /* DO WHILE */
/*    IF (INCHR = 'D') OR (INCHR = 64H) THEN DO;
/*       /* DISPLAY COMMAND SECTION */
/*       CALL IN$DNUM(@ADDR.BASE);
/*       CALL OUT$CHAR(':');
/*       CALL IN$DNUM(@ADDR.OFFSET);
/*       PTR2 = @ADDR;
/*       PTR = PTR3;
/*       /* CONTENTS SHOULD NOW BE SET */
/*       DO WHILE (INCHR<>CR) AND (INCHR<>23H);
/*          CALL IN$CHAR(@INCHR);
/*       END; /* DO WHILE */
/*       IF INCHR = CR THEN DO;
/*          CALL OUT$CHAR('-');
/*          CALL OUT$NUM(CONTENTS);
/*          CALL OUT$CHAR(CR);
/*          CALL OUT$CHAR(LF);
/*       END; /* IF NORMAL 1 ADDR DISPLAY */
/*       IF INCHR = 23H THEN DO;
/*          COUNT = 0;
/*          CALL OUT$CHAR('#');
/*          CALL IN$NUM(@QUANTITY);
/*          DO WHILE QUANTITY > 0;
/*             CALL OUT$CHAR(CR);
/*             CALL OUT$CHAR(LF);
/*             CALL OUT$DNUM(ADDR.BASE);
/*             CALL OUT$CHAR(':');
/*             CALL OUT$DNUM(ADDR.OFFSET);
/*             LINECOMPLETE = FALSE;
/*             DO WHILE LINECOMPLETE = FALSE;
/*                CALL OUT$CHAR(' ');
/*                CALL OUT$NUM(CONTENTS);
/*                ADDR.OFFSET = ADDR.OFFSET + 1;
/*                PTR = PTR3;
/*                QUANTITY = QUANTITY - 1;
/*                IF ((ADDR.OFFSET AND 000FH)=0) OR
/*                   (QUANTITY = 0) THEN LINECOMPLETE=TRUE;
/*             END; /* DO WHILE LINE NOT COMPLETE */
/*          END; /* DO WHILE QUANTITY */
/*       END; /* IF MULTI ADDR DISPLAY */
/*    END; /* DISPLAY COMMAND SECTION */
/*    IF (INCHR='S') OR (INCHR=73H) THEN DO;
/*       /* SUBSTITUTE COMMAND SECTION */
/*       CALL IN$DNUM(@ADDR.BASE);
/*       CALL OUT$CHAR(':');
/*       CALL IN$DNUM(@ADDR.OFFSET);
/*       CALL OUT$CHAR('-');

/*       PTR2 = @ADDR;
```

```
/*        PTR = PTR3;
/*        /* CURRENT CONTENTS SHOULD NOW BE AVAILABLE */
/*        CALL OUT$NUM(CONTENTS);
/*        LOOP2 = TRUE;
/*        DO WHILE LOOP2 = TRUE;
/*           DO WHILE (INCHR<>',')AND(INCHR<>' ')
/*              AND(INCHR<>CR);
/*              CALL IN$CHAR(@INCHR);
/*           END;
/*           IF (INCHR = CR) THEN LOOP2 = FALSE;
/*           IF (INCHR = ',') THEN DO;
/*              /* SKIP THIS ADDR AND GO TO NEXT FOR SUB */
/*              CALL OUT$CHAR(CR);
/*              CALL OUT$CHAR(LF);
/*              ADDR.OFFSET = ADDR.OFFSET + 1;
/*              PTR = PTR3;
/*              CALL OUT$DNUM(ADDR.BASE);
/*              CALL OUT$CHAR(':');
/*              CALL OUT$DNUM(ADDR.OFFSET);
/*              CALL OUT$CHAR('-');
/*              CALL OUT$NUM(CONTENTS);
/*           END;  /* IF SKIP FOR NEXT SUB */
/*           IF (INCHR = ' ') THEN DO;
/*              CALL OUT$CHAR(' ');
/*              CALL IN$NUM(@CONTENTS);
/*              DO WHILE (INCHR<>CR)AND(INCHR<>'.');
/*                 CALL IN$CHAR(@INCHR);
/*              END;
/*              IF (INCHR = CR) THEN LOOP2 = FALSE;
/*              IF (INCHR = ',') THEN DO;
/*                 CALL OUT$CHAR(',');
/*                 ADDR.OFFSET = ADDR.OFFSET + 1;
/*                 PTR = PTR3;
/*                 CALL OUT$CHAR(CR);
/*                 CALL OUT$CHAR(LF);
/*                 CALL OUT$DNUM(ADDR.BASE);
/*                 CALL OUT$CHAR(':');
/*                 CALL OUT$DNUM(ADDR.OFFSET);
/*                 CALL OUT$CHAR('-');
/*                 CALL OUT$NUM(CONTENTS);
/*              END; /* IF GO TO NEXT ADDR */
/*           END; /* IF CHANGE CONTENTS */
/*           INCHR = 'X'; /* REINITIALIZE CMD */
/*        END; /* LOOP, CONTINUOUS SUB CMD */
/*     END; /* SUBSTITUTE COMMAND SECTION */

/*     IF (INCHR='E') OR (INCHR=65H) THEN DO;
/*        /* FIND OUT WHICH VPS IS RUNNING 'ME' */
/*        INDEX = RET$VP;
/*        /* NOW BLOCK MYSELF */
/*        DISABLE;
/*        PRDS.LAST$RUN = INDEX;
```

```
/*       VPM(INDEX).STATE = WAITING;
/*       CALL VPSCHEDULER; /* NO RETURN */
/*    END; /* IF */
/* GO TO LOOP;
/* END; /* MONITOR PROCESS */
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/


/*0955*************************************************************/
/*************************************************************/
/*          STARTING POINT OF THE OPERATING SYSTEM           */
/*-----------------------------------------------------------*/
/* ROUTINE INITIALIZES THE OS AND IS NOT REPEATED.           */
/*************************************************************/
/*************************************************************/

/* TO INITIALIZE THE PRDS TABLE FOR THIS CPU */
DECLARE CPU$PTR POINTER DATA(@PRDS.CPU$NUMBER),
        ZZ BYTE;

DISABLE;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* CALL OUT$LINE(@MSG12);
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

/* INITIALIZE   P P I   AND   P I C */
OUTPUT(PORT$CE) = 0C0H;/* PPI - MICROPOLIS + MCORTEX */
OUTPUT(PORT$C0) = 13H;  /* PIC - ICW1 - EDGE TRIGGERED */
OUTPUT(PORT$C2) = 40H;/* PIC - ICW2 -VECTOR TABLE ADDRESS */
OUTPUT(PORT$C2) = 0FH;/* PIC - ICW4 -MCS86 MODE. AUTO EOI */
OUTPUT(PORT$C2) = 0AFH; /*PIC - MASK ALLOWING INT. 4 & 6 */

/* ESTABLISH UNIQUE SEQUENTIAL NUMBER FOR THIS CPU */
/* SET GLOBAL$LOCK */
DO WHILE LOCK$SET(@GLOBAL$LOCK,119);  END;
PRDS.CPU$NUMBER = CPU$INIT;
CPU$INIT = CPU$INIT + 1;

/* RELEASE GLOBAL LOCK */
GLOBAL$LOCK = 0;

/*  SET UP INITIAL START AND END FOR PROC TABLE */
PRDS.VP$START = 0;
DO ZZ = 1 TO PRDS.CPU$NUMBER;
   PRDS.VP$START = PRDS.VP$START + MAX$VPS$CPU;
END;
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
```

```
PRDS.VP$END = PRDS.VP$START + 1;
PRDS.VPS$PER$CPU = 2;
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* PRDS.VP$END = PRDS.VP$START + 2;
/* PRDS.VPS$PER$CPU = 3;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

/* INITIALIZE THE VP MAP FOR IDLE AND INIT PROC */
/* AND MONITOR PROCESS */
VPM(PRDS.VP$START).VP$ID = 255;
VPM(PRDS.VP$START).STATE = 1;
VPM(PRDS.VP$START).VP$PRIORITY = 0;
VPM(PRDS.VP$START).EVC$THREAD = 255;
VPM(PRDS.VP$START).EVC$AW$VALUE = 0;
VPM(PRDS.VP$START).SP$REG = 60H;
VPM(PRDS.VP$START).SS$REG = INIT$STACK$SEG;
VPM(PRDS.VP$START+1).VP$ID = 255;
VPM(PRDS.VP$START+1).STATE = 1;
VPM(PRDS.VP$START+1).VP$PRIORITY = 255;
VPM(PRDS.VP$START+1).EVC$THREAD = 255;
VPM(PRDS.VP$START+1).EVC$AW$VALUE = 0;
VPM(PRDS.VP$START+1).SP$REG = 60H;
VPM(PRDS.VP$START+1).SS$REG = IDLE$STACK$SEG;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* VPM(PRDS.VP$START+2).VP$ID = 0FEH;
/* VPM(PRDS.VP$START+2).STATE = 7;
/* VPM(PRDS.VP$START+2).VP$PRIORITY = 0;
/* VPM(PRDS.VP$START+2).EVC$THREAD = 255;
/* VPM(PRDS.VP$START+2).EVC$AW$VALUE = 0;
/* VPM(PRDS.VP$START+2).SP$REG = 60H;
/* VPM(PRDS.VP$START+2).SS$REG = MONITOR$STACK$SEG;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

NR$RPS = NR$RPS + 1;

/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
NR$VPS(PRDS.CPU$NUMBER) = 2;
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/* NR$VPS(PRDS.CPU$NUMBER) = 3;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

HDW$INT$FLAG( PRDS.CPU$NUMBER ) = 0 ;
ENABLE;
```

```
PRDS.LAST$RUN = 255; /* INDICATE START ENTRY TO SCHEDULER */
CALL VPSCHEDULER;          /* - - NO RETURN */

/*****************************************************************/
/*****************************************************************/

END;  /* L1$MODULE */

/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
```

```
/*** MCORTEX ***** MCORTEX ***** MCORTEX ***** MCORTEX ***/

.F1:LOC86 KORE.LNK ADDRESSES(SEGMENTS(&
STACK(0C780H),&
INITMOD_CODE(04390E),&
GLOBALMODULE_DATA(0E7942H)))&
SEGSIZE(STACK(75H))&
RESERVE(0H TO 0BAFFH)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:  (NO NAME)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:  INITMOD_CODE

SYMBOL TABLE OF MODULE L1MODULE
READ FROM FILE KORE.LNK
WRITTEN TO FILE :F0:KORE

BASE   OFFSET TYPE SYMBOL       BASE   OFFSET TYPE SYMBOL

0C6AH  0000H PUB PRDS           0BB0H  0380H PUB IDLEPROC
0BB0H  0302H PUB LOCATESEQ      0BB0H  0284H PUB LOCATEEVC
0BB0H  0208H PUB GETWORK        0BB0H  01E3H PUB GETSP
0B30H  01AEH PUB SAVECONTEXT    0BB0H  0185H PUB RDYTHISVP
0BB0H  013AH PUB RETVP          0BEBH  0685H PUB CREATEPROC
0BFBH  062AH PUB TICKET         0BEBH  05C7H PUB CREATESEQ
0BEBH  03CFH PUB PREEMPT        0BEBH  0228H PUB ADVANCE
0BEBH  0178H PUB AWAIT          0BE3H  0127H PUB READ
0BEBH  00FFH PUB CREATEEVC      0BE3H  002EH PUB GATEKEEPER
0C6BH  0000H PUB VPSCHEDULER    0C6BH  0033H PUB INTVEC
0439H  0002H PUB INITIALPROC    E794H  0192H PUB VPM
E794H  0593H PUB SEQTABLE       E794H  0592H PUB SEQUENCERS
E794H  0591H PUB CPUINIT        E794H  0002H PUB EVCTBL
E794H  0590H PUB EVENTS         E794H  0586H PUB HDWINTFLAG
E794H  057CH PUB NRVPS          E794H  057BH PUB NRRPS
E794H  057AH PUB GLOBALLOCK
```

```
MEMORY MAP OF MODULE L1MODULE
READ FROM FILE KORE.LNK
WRITTEN TO FILE :F2:KORE

MODULE START ADDRESS  PARAGRAPH = 0FB0H  OFFSET = 0030H
SEGMENT MAP

START     STOP      LENGTH ALIGN NAME                CLASS

0011CH    00113H    0004H   A    (ABSOLUTE)
0439CH    043A9H    001AH   W    INITMOD_CODE        CODE
0FB0CH    0BFF3H    03B4H   W    L1MODULE_CODE       CODE
0BFB4H    0C69FFH   07ECH   W    L2MODULE_CODE       CODE
0C6A0H    0C6A2H    0003H   W    GLOBALMODULE_C      CODE
                              -ODE
0C6A0H    0C6A7H    0008H   W    L1MODULE_DATA       DATA
0C6A8H    0C6A8H    0000H   W    L2MODULE_DATA       DATA
0C6A9H    0C6A9H    0001H   W    INITMOD_DATA        DATA
0C6B0H    0C6B0H    0000H   G    ??SEG
0C6B0H    0C746H    0097H   G    SCHEDULER
0C7B0H    0C7F4H    0075H   W    STACK               STACK
0C800H    0C879H    007AH   A    (ABSOLUTE)
0C880H    0C8F9H    007AH   A    (ABSOLUTE)
F7942H    E7FFEH    06BDH   W    GLOBALMODULE_D      DATA
                              -ATA
E8000H    E8000H    0000H   W    MEMORY              MEMORY
```

```
ISIS-II MCS-86 LOCATER, V1.1 INVOKED BY:
:F1:LOC86 KORE.LNK ADDRESSES(SEGMENTS(&
STACK(0C480H),&
INITMOD_CODE(04390H),&
GLOBALMODULE_DATA(0E7942H)))&
SEGSIZE(STACK(75H))&
RESERVE(0H TO 0ABFFH)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:   (NO NAME)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:   INITMOD_CODE

SYMBOL TABLE OF MODULE L1MODULE
READ FROM FILE KORE.LNK
WRITTEN TO FILE :F0:KORE

BASE   OFFSET TYPE SYMBOL          BASE   OFFSET TYPE SYMBOL

0C2DH  00CAH PUB PRDS              0AC0H  04E6H PUB MONITOR$PROC
0AC0H  049CH PUB IDLE$PROC         0AC0H  03FDH PUB LOCATE$SEQ
0AC0H  035EH PUB LOCATE$EVC        0AC0H  0293H PUB GET$WORK
0AC0H  026BH PUB GET$SP            0AC0H  0236H PUB SAVE$CONTEXT
0AC0H  01DEH PUB RDY$THIS$VP       0AC0H  0165H PUB RET$VP
0B4AH  0C06H PUB OUTHEX            0B4AH  0B01H PUB INHEX
0B4AH  0AB1H PUB SENDCHAR          0B4AH  0A8EH PUB RECVCHAR
0B4AH  0A62H PUB OUTDNUM           0B4AH  0A29H PUB INDNUM
0B4AH  0A11H PUB OUTNUM            0B4AH  09C2H PUB OUTLINE
0B4AH  09AAH PUB OUTCHAR           0B4AH  098EH PUB INNUM
0B4AH  0974H PUB INCHAR            0B4AH  0804H PUB CREATE$PROC
0B4AH  0798H PUB TICKET            0B4AH  0712H PUB CREATE$SEQ
0B4AH  04F9H PUB PREEMPT           0B4AH  033CH PUB ADVANCE
0B4AH  0281H PUB AWAIT             0B4AH  020DH PUB READ
0B4AH  0182H PUB CREATE$EVC        0B4AH  0062H PUB GATEKEEPER
0C31H  0070H PUB VP$SCHEDULER      0C31H  0033H PUB INTVEC
0439H  0002H PUB INITIAL$PROC      E794H  0192H PUB VPM
E794H  0593H PUB SEQTABLE          E794H  0592H PUB SEQUENCERS
E794H  0591H PUB CPUINIT           E794H  0002H PUB EVCTBL
E794H  0590H PUB EVENTS            E794H  0586H PUB HDWINTFLAG
E794H  057CH PUB NRVPS             E794H  057BH PUB NRRPS
E794H  057AH PUB GLOBALLOCK
```

```
MEMORY MAP OF MODULE L1MODULE
READ FROM FILE KORE.LNK
WRITTEN TO FILE :F0:KORE

MODULE START ADDRESS  PARAGRAPH = 0AC0H  OFFSET = 0030H
SEGMENT MAP

START     STOP      LENGTH ALIGN NAME              CLASS

0011EH    00113H    0004H   A    (ABSOLUTE)
04390H    043B4H    0025H   W    INITMOD_CODE      CODE
0AC00H    0B4A6H    08A7H   W    L1MODULE_CODE     CODE
0B4A8H    0C0D9H    0C32H   W    L2MODULE_CODE     CODE
0C0DAH    0C0DAH    0000H   W    GLOBALMODULE_C    CODE
                              -ODE
0C0DAH    0C2CCH    0133H   W    L1MODULE_DATA     DATA
0C2CEH    0C2F1H    00E4H   W    L2MODULE_DATA     DATA
0C2F2H    0C30EH    001EH   W    INITMOD_DATA      DATA
0C310H    0C310H    0000H   G    ??SEG
0C310H    0C3A6H    0097H   G    SCHEDULER
0C480H    0C4F4H    0075H   W    STACK             STACK
0C500H    0C579H    007AH   A    (ABSOLUTE)
0C580H    0C5F9H    007AH   A    (ABSOLUTE)
0C600H    0C679H    007AH   A    (ABSOLUTE)
E7942H    E7FFEH    06BDH   W    GLOBALMODULE_D    DATA
                              -ATA
F8000H    F8002H    0002H   W    MEMORY            MEMORY
```

APPENDIX I

SCHEDULER & INTERRUPT HANDLER SOURCE CODE

The ASM86 code in file: SCHED.ASM is part of LEVEL I. No special attributes are required for the assembler. This module is linked into file: KORE.LNK, and its memory map is included in the map for KORE. KORE is the development system version of the file KORE.OPS loaded by MCOPTEX.CMD under the CP/M-86 operating system.

```
;*0027*****************************************************
;*   SCHEDULER      ASM FILE                ROWE 6-22-84    *
;*-------------------------------------------------------*
;* THE FOLLOWING ARE THE EXTERNAL PLM86 PROCEDURES CALLED  *
;* BY THIS MODULE.                                         *

    EXTRN SAVECONTEXT:FAR
    EXTRN GETSP:FAR
    EXTRN GETWORK:FAR
    EXTRN RDYTHISVP:FAR
    EXTRN PRDS:BYTE
    EXTRN HDWINTFLAG:BYTE
    EXTRN GLOBALLOCK:BYTE

SCHEDULER SEGMENT

    PUBLIC VPSCHEDULER
    PUBLIC INTVEC

VPSCHEDULER PROC FAR

    ASSUME CS:SCHEDULER
    ASSUME DS:NOTHING
    ASSUME SS:NOTHING
    ASSUME ES:NOTHING

    ; ENTRY POINT FOR A CALL TO SCHEDULER

    CLI
    PUSH DS
    MOV CX,0H

    ;SWAP VIRTUAL PROCESSORS.  THIS IS DONE BY SAVING THE
    ;STACK BASE POINTER AND THE RETURN TYPE FLAG ON THE
    ;STACK, AND BY SAVING THE STACK SEGMENT AND STACK
    ;POINTER IN THE VIRTUAL PROCESSOR MAP.

    INTJOIN: PUSH BP          ;SAVE "CURRENT" STACK BASE
      PUSH CX                 ;SAVE 'CURRENT' IRET_IND FLAG

      MOV AX,SP
      PUSH AX                 ;SET UP SAVE$CONTEXT PARAMETERS
      PUSH SS                 ;SET UP SAVE$CONTEXT PARAMETERS
      CALL SAVECONTEXT

      CALL GETWORK            ;GET NEW STACK SEGMENT
      PUSH AX                 ;TEMPORARY SAVE OF STACK SEGMENT
      CALL GETSP              ;GET NEW STACK POINTER
      POP SS                  ;INSTALL NEW STACK SEGMENT
      MOV SP,AX               ;INSTALL NEW STACK POINTER

      ;SWAP VIRTUAL PROCESSOR CONTEXT COMPLETE AT THIS POINT
```

```
      ;NOW OPERATING IN NEWLY SELECTED PROCESS STACK

       POP CX                          ;GET IRET_IND FLAG
       POP BP                          ;INSTALL NEW STACK BASE

       ;  CHECK FOR RETURN TYPE, NORMAL OR INTERRUPT

       CMP CX,77H
       JZ  INTRET

    NORM_RET: POP DS
       ; UNLOCK GLOBAL$LOCK
       MOV  AX,SEG GLOBALLOCK
       MOV  ES, AX
       MOV  ES:GLOBALLOCK,0

       STI
       RET

VPSCHEDULER ENDP

;*****************************************************************************


;*****************************************************************************
;*  INTERRUPT HANDLER                                                        *
;*                                                                           *

INTERRUPT_HANDLER  PROC NEAR

    ASSUME CS:SCHEDULER
    ASSUME DS:NOTHING
    ASSUME SS:NOTHING
    ASSUME ES:NOTHING

INTVEC: CLI
    PUSH ES     ; SAVE NEEDED REGs TO TEST INTERRUPT FLAG
    PUSH BX
    PUSH AX
    PUSH CX
    CALL HARDWARE_INT_FLAG
    MOV  AL,0
    XCHG  AL,ES:HDWINTFLAG[BX]
    CMP  AL,77H              ; IS INT FLAG ON ?
    JZ   PUSH_REST_REGS      ; IF 'YES' SAVE REST REGs
    POP  CX                  ; IF 'NOT' RESUME PREVIOUS
    POP  AX                  ; EXECUTION POINT
    POP  BX
    POP  ES
    STI
    IRET
```

```
PUSH_REST_REGS: PUSH DX    ; FLAG WAS ON SO NEED
   PUSH DS                 ; RE-SECHEDULE
   PUSH SI
   PUSH DI
   MOV AX,SEG GLOBALLOCK
   MOV  ES, AX
CK: MOV AL,119             ; LOCK GLOBAL LOCK
   LOCK XCHG ES:GLOBALLOCK,AL
   TEST AL,AL
   JNZ CK

   CALL RDYTHISVP

   MOV CX,77H           ; JUMP TO SCHEDULER
   JMP  INTJOIN

INTRET: POP DI
   POP SI                  ; RETURN FOR
   POP DS                  ; PROCESS WHICH
   POP DX                  ; HAD PREVIOUSLY
   POP CX                  ; BEEN INTERRUPTED
        ; UNLOCK GLOBAL$LOCK
   MOV  AX,SEG GLOBALLOCK
   MOV  ES, AX
   MOV  ES:GLOBALLOCK,0

   POP AX
   POP BX
   POP ES
   STI
   IRET

INTERRUPT_HANDLER ENDP

;*****************************************************************

;*****************************************************************
;*    HARDWARE INTERRUPT FLAG                                    *
;*                                                               *

HARDWARE_INT_FLAG  PROC  NEAR

   ASSUME CS:SCHEDULER
   ASSUME DS:NOTHING
   ASSUME SS:NOTHING
   ASSUME ES:NOTHING
HDW_FLAG: MOV  AX,SEG PRDS

   MOV  ES, AX
   MOV  BX,0H
```

```
    MOV   CL,ES:PRDS[BX]     ;GET CPU #
    MOV   CH,0               ; RETURN IN BX
    MOV   BX,CX
    MOV   AX,SEG HDWINTFLAG  ;SET UP HDW$INT$FLAG
    MOV   ES, AX             ;       SEGMENT
    RET                      ; RETURN IN ES REG

    HARDWARE_INT_FLAG  ENDP

    SCHEDULER ENDS
END
```

APPENDIX J

GLOBAL DATA BASE AND INITIAL PROCESS CODE

Two files are presented here: GLOBAL.SRC and INITK.SRC. They are both separately compiled with the LARGE attribute. They are linked into the file: KORE.LNK. They are represented in the memory map for KORE located at the end of Appendix H. INITK will be overwritten by the users initialization process.

```
/*****************************************************************/
/*****************************************************************/
/*0009***********************************************************/
/*  FILE:           GLOBAL.SRC
    VERSION:        ROWE 6-22-84
    PROCEDURES
        DEFINED:    NONE

    REMARKS:THIS MODULE CONTAINS DECLARATIONS FOR ALL THE
            GLOBAL DATA THAT RESIDES IN SHARED COMMON
            MEMORY.  IT'S LOCATED THERE BY THE LOCATE COM-
            MAND AND BY SPECIFYING THAT THE
            GLOBAL$MODULE_DATA SEGMENT BE LOCATED AT SOME
            ABSOLUTE ADDRESS.
                                                            */
/*****************************************************************/

GLOBAL$MODULE:  DO;

/*****************************************************************/
/*****************************************************************/
/*  THE FOLLOWING THREE LITERAL DECLARATIONS ARE ALSO          */
/*  GIVEN IN THE LEVEL1 & LEVEL2 MODULES OF THE OPERATING      */
/*  SYSTEM.  A CHANGE HERE WOULD HAVE TO BE REFLECTED IN       */
/*  THOSE MODULES ALSO.                                        */

DECLARE
   MAX$CPU                    LITERALLY  '10',
   MAX$VPS$CPU                LITERALLY  '10',
   MAX$CPU$$$$MAX$VPS$CPU LITERALLY '100';

DECLARE
   GLOBAL$LOCK BYTE PUBLIC INITIAL(0);

/*  THIS SHOULD REFLECT THE MAX$CPU ABOVE */
DECLARE
       NR$RPS             BYTE PUBLIC INITIAL(0),
       NR$VPS(MAX$CPU) BYTE PUBLIC
                          INITIAL(0,0,0,0,0,0,0,0,0,0);

DECLARE HDW$INT$FLAG(MAX$CPU) BYTE PUBLIC;


DECLARE EVENTS  BYTE  PUBLIC INITIAL(1);

DECLARE EVC$TBL(100) STRUCTURE
        (EVC$NAME     BYTE,
         VALUE        WORD,
         THREAD       BYTE)  PUBLIC
                             INITIAL(0FEH,0,255);
   /* EVC "FE" IS RESERVED FOR THE OP SYS  */
```

```
DECLARE CPU$INIT BYTE PUBLIC INITIAL(0);

DECLARE SEQUENCERS      BYTE  PUBLIC INITIAL(0);

DECLARE SEQ$TABLE(100) STRUCTURE
        (SEQ$NAME       BYTE,
         SEQ$VALUE      WORD) PUBLIC;

DECLARE VPM( MAX$CPU$$$$MAX$VPS$CPU ) STRUCTURE
        (VP$ID                  BYTE,
         VP$STATE               BYTE,
         VP$PRIORITY            BYTE,
         EVC$TEREAD             BYTE,
         EVC$AW$VALUE           WORD,
         SP$REG                 WORD,
         SS$REG                 WORD)  PUBLIC;


 END; /* MODULE */

/*****************************************************************/
```

```
/*************************************************************/
/*   INITX      MODULE                          ROWE 6-22-84 */
/*-----------------------------------------------------------*/
/* THE CODE SEGMENT OF THIS MODULE IS WHAT RESERVES SPACE    */
/* BY THE OS FOR THE USER INITIAL PROCESS.  THIS IS          */
/* EXECUTABLE IN IT'S OWN RIGHT.  THUS IF THE USER DOES      */
/* NOT PROVIDE AN INITIAL PROCESS THIS ONE WILL EXECUTE,     */
/* BLOCK ITSELF, AND IDLE THE CPU.  THE ADDRESS OF THE       */
/* INITIAL CODE SEGMENT IS PROVIDED TO LEVEL1 AND IT IS      */
/* REFLECTED IN THE PLM LOCATE COMMAND.  THE ADDRESSES       */
/* PROVIDED MUST AGREE.  THIS PROCESS HAS THE HIGHEST        */
/* PRIORITY AND WILL ALWAYS BE SCHEDULED FIRST BY THE        */
/* SCHEDULER.                                                */
/*-----------------------------------------------------------*/
/* CALLS MADE TO:   AWAIT                                    */
/*************************************************************/
INIT$MOD: DO;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*    DECLARE
/*       MSG13(*) BYTE INITIAL(10,'ENTERING INITIAL PROCESS ',
/*                                13,10,'%');
/*    OUT$LINE: PROCEDURE( PTR ) EXTERNAL;
/*       DECLARE PTR POINTER;
/*    END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
   AWAIT:  PROCEDURE( NAME, VALUE ) EXTERNAL;
      DECLARE NAME BYTE, VALUE WORD;
   END;

   INITIAL$PROC: PROCEDURE PUBLIC;

      DECLARE I BYTE;
      /* AFTER INITIALIZATION THIS PROCESS BLOCKS      */
      /* ITSELF TO ALLOW THE NEWLY CREATED PROCESSES   */
      /* TO BE SCHEDULED.                              */
      /* THIS AREA SHOULD BE WRITTEN OVER BY USER INIT */
      /* PROCEDURE MODULE.                             */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*    CALL OUT$LINE(@MSG13);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

      CALL AWAIT( 0FEH, 1);

   END;    /* INITIAL$PROC */
END;  /* INIT$MOD */
```

LIST OF REFERENCES

1. Cox, F. R., A Real-Time, Distributed Operating System for a Multiple Computer System, M.S. Thesis, Naval Postgraduate School, December 1981.

2. Klienfelter, S.G., Implementation of a Real-Time, Distributed Operating System for a Multiple Computer System, M.S. Thesis, Naval Postgraduate School, June 1982.

3. INTEL Corperation, The 8086 Family User's Manual, October 1979.

4. INTEL Corperation, iSBC 86/12 Single Board Computer Hardware Reference Manual, 1976.

5. Perry, M. L., Logic Design of a Shared Disk System in a Multi-Micro-Computer Environment, M.S. Thesis, Naval Postgraduate School, June 1983.

6. Digital Research, CP/M-86 Programmer's Guide, 1981.

7. Digital Research, CP/M-86 System Guide, 1981.

8. Digital Research, PL/I Language Reference Manual, 1982.

9. Digital Research, Programmer's Utilities Guide for the CP/M-86 Family of Operating Systems, 1982.

10. Digital Research, PL/I Language Programmer's Guide, 1982.

11. INTEL Corperation, 8086 Family Utilities User's Guide for 8080/8085 Based Development Systems, 1980.

12. INTEL Corperation, ISIS-II PL/M-86 Compiler Operator's Manual, 1978.

13. INTEL Corperation, ISIS-II User's Guide, 1978.

14. INTEL Corperation, MCS-86 Assembler Operating Instructions for ISIS-II Users, 1978.

15. INTEL Corperation, MCS-86 Macro Assembly Language Reference Manual, 1979.

16. INTEL Corperation, PL/M-86 Programming Manual for 8080/8085 Based Development Systems, 1980.

INITIAL DISTRIBUTION LIST

		No. of copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22341	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
4.	Dr. M. L. Cotton, Code 62Cc Department of Electrical Engineering Naval Postgraduate School Monterey, California 93943	1
5.	Dr. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943	3
6.	Commanding Officer VXF-6 ATTN: LT Willis R. Rowe Point Mugu, California 93042	1
7.	Daniel Green, Code 20F Naval Surface Weapons Center Dahlgren, Virginia 22449	1
8.	CAPT J. Donegan, USN PMS 400B5 Naval Sea Systems Command Washington, D.C. 20362	1
9.	RCA AEGIS Data Repository RCA Corperation Government Systems Division Mail Stop 127-327 Moorestown, New Jersey 08057	1

10. Library (code E33-05) 1
Naval Surface Warfare Center
Dahlgren, Virginia 22449

11. Dr. M. J. Gralia 1
Applied Physics Laboratory
Johns Hopkins Road
Laurel, Maryland 20707

12. Dana Small 1
Code 8242, NOSC
San Diego, California 92152

210581

Thesis
R8186 Rowe
c.1 Adaptation of MCORTEX to the AEGIS simulation environment.